The EGRBAC Model for Smart Home IoT

Safwa Ameer
Institute for Cyber Security and CREST C-SPECC
Department of Computer Science
The University of Texas at San Antonio
San Antonio, USA
Safwa.Ameer@my.utsa.edu

James Benson
Institute for Cyber Security and CREST C-SPECC
Department of Computer Science
The University of Texas at San Antonio
San Antonio, USA
James.Benson@utsa.edu

Ravi Sandhu
Institute for Cyber Security and CREST C-SPECC
Department of Computer Science
The University of Texas at San Antonio
San Antonio, USA
ravi.sandhu@utsa.edu

Abstract—The Internet of Things (IoT) is enabling smart houses, where multiple users with complex social relationships interact with smart devices. This requires sophisticated access control specification and enforcement models, that are currently lacking. In this paper, we introduce the extended generalized role based access control (EGRBAC) model for smart home IoT. We provide a formal definition for EGRBAC and illustrate its features with a use case. A proof-of-concept demonstration utilizing AWS-IoT Greengrass is discussed in the appendix. EGRBAC is a first step in developing a comprehensive family of access control models for smart home IoT.

Index Terms—Access Control, Smart home IoT, RBAC

I. INTRODUCTION AND MOTIVATION

The smart home is one of the most popular domains for deploying the Internet of Things (IoT), envisioned as a global network of machines and devices capable of interacting with each other [1]. Nevertheless, surprisingly little attention has been paid to access control policy specification and enforcement in home IoT [2].

Authorization issues in home IoT are significantly different from traditional domains in three main aspects. First, we have many users who use the same device, for example a smart door lock. Second, house occupants usually have complex social relationships, which introduces a new threat model, such as an annoying child trying to control the smart light in a sibling’s room, or a current or ex-partner trying to abuse one or all house residents [2], [3]. Another major characteristic of IoT devices is that the majority lack a screen and keyboard making them hands free for convenience while making access control more challenging. These characteristics suggest the need for a dynamic and fine-grained access control model for smart home IoT, where users and resources are constrained [4].

In this paper we describe our first access control model for smart home IoT. Why focus on the home rather than general IoT? We believe that smart homes provide a rich yet scoped environment where we have a limited number of users who want to access a limited number of shared constrained smart things with different privileges. Such scoping is useful to develop an initial set of models. In future these scoped models can be adapted and evolved to address the access control requirements of other IoT domains, such as a smart office, a smart classroom or a smart city.

Our model is inspired by the early work of Covington et al [5] who extended role-based access control (RBAC) to the smart home environment in a model called Generalized RBAC (GRBAC). We call our model the extended GRBAC (EGRBAC) model. EGRBAC, like GRBAC, focuses on user to device (U-D) interaction, leaving device to device (D-D) for future work. One goal of EGRBAC is to investigate the limitations of applying RBAC concepts in home IoT. In future work we plan to develop models that incorporate concepts of attribute-based access control (ABAC) and demonstrate their benefits relative to EGRBAC.

The paper is organized as follows. Section II identifies desirable criteria for smart home IoT access control models. An analysis and review of related work is given in Section III. Section IV provides an overview of GRBAC [5] and of the architecture that we adopt to enforce EGRBAC. Section V describes our threat model. In Section VI, we introduce EGRBAC along with a use case scenario, analyze EGRBAC against our proposed criteria and discuss the limitations of EGRBAC. A proof-of-concept demonstration is discussed in Section A. Section VIII concludes the paper.

II. CRITERIA FOR HOME IoT ACCESS CONTROL

We begin by proposing at least the following criteria for home IoT access control models (whether U-D, D-D or both), based in part on He et al [2] and Ouddah et al [4].

1) The model should be dynamic so as to capture environment and object contextual information.
2) The model should be fine-grained so that a subset of the

This work is partially supported by NSF CREST Grant HRD1736209.
particular, many researchers have conducted studies of IoT security challenges and security design issues in the home environment. In Section VI we justify the evaluation of EGRBAC according to the characteristics in this table.

### III. RELATED WORK

Smart home IoT has been extensively studied by security experts. Many researchers have focused on identifying IoT security and privacy vulnerabilities [20]–[27]. Moreover, to analyze IoT security challenges and security design issues in particular, many researchers have conducted studies of IoT frameworks (e.g. [20], [24], [28]–[30]). One of the critical security services in IoT that mostly all researchers agree upon is access control. Ouaddah et al [4] have extensively investigated access control in IoT environments. The IoT access control models in Table I are based on RBAC [31]–[33], ABAC [34], [35], UCON [36] or CapBAC [4]. Our assessment of these models with respect to the above criteria is summarized in the table. Going beyond the models in this table, some approaches based on blockchain technology have been proposed (e.g. [37]–[39]). However, as [38] described, the blockchain technology has some technical characteristics that could limit its applicability such as, cryptocurrency fees and slow processing time. The authors in [4], [40], [41] provide surveys on additional models beyond Table I such as focussed D-D only. However, none of them meet all the criteria of the table.

#### IV. BACKGROUND

##### A. The GRBAC Model

Covington et al introduced the Generalized Role-Based Access Control (GRBAC) model [5]. In addition to the usual concept of User Roles, GRBAC incorporates the notion of Object Roles and Environment Roles. A user role is analogous to the traditional RBAC role. An object role is defined as the properties of the resources in the system, such as images, source code, streaming videos, devices. An environment role is defined as the environment state during access. Covington et al [43] subsequently described an architecture to support environment roles activation according to the current environment conditions. They also provided a high level but incomplete formal definition of environment role based access control model, building upon [8]. They did not formalize the object role part of GRBAC. In this paper, we provide a more fine grained model with a detailed formalization. However, we used devices instead of objects since it is more appropriate to smart homes.

##### B. IOT Based Smart Home Architecture

The smart home IoT architecture that we adopted for EGRBAC enforcement was introduced by Geneiatakis et al [42]. It is illustrated in Fig. 1. The IoT devices are connected to a corresponding hub and are not directly accessed by other devices or by users. The intermediate hub is responsible for functionality of a device can be authorized rather than all-or-nothing access to the device.

3) Smart things in homes are usually limited in term of computational power, and storage. Furthermore, a generic interoperability standard among IoT devices is still missing. Accordingly, the model should be suitable for constrained smart home devices. In other words, it should not require extensive computation or communication by those constrained devices.

4) The model should be constructed specifically for smart home IoT, or otherwise be interpreted for the smart home domain such as by appropriate use cases, to ensure that the model is suitable for smart home different specifications such as, social relationships between house members, cost effectiveness, usability, and so on.

5) The model should be demonstrated in a proof-of-concept to be credible using commercially available technology with necessary enhancements.

6) The model should have a formal definition, so that there is a precise and rigorous specification of the intended behavior.

We analyzed IoT access control models proposed in the literature based on these six characteristics. A summary of the analysis is provided in Table I. In this table we only included access control models that govern device to device access, since this is the scope of our model. From the table we can notice that except for our model (summarized in the first row), no model satisfies all desired characteristics. Furthermore, surprisingly, except for EGRBAC and GRBAC, no model was designed or interpreted explicitly for smart home environment. In Section VI we justify the evaluation of EGRBAC according to the characteristics in this table.
providing Internet connectivity, since the majority of commercial sensors do not provide direct Internet connectivity. The communication between the smart hub and the IoT devices is usually wireless, through different protocols such as Zigbee, Z-Wave and WiFi. In order to connect the smart IoT devices optionally, to the outside world, the hub is connected to the home’s routers via an Ethernet or a Wi-Fi interface. In general there are two types of access. In local access users directly interact with the IoT devices through the connectivity services provided by the hub. In remote access users access IoT devices via cloud services, which in turn communicate with the smart hub via the Internet to access these devices.

V. Threat Model

In smart houses we recognize two types of adversaries [2]. First, an outside hacker who is trying to get digital or physical access to the house by exploiting system vulnerabilities. Second the household members themselves. These are insiders who have legitimate digital and physical access to the house, such as family members, guests, and workers. The motivation for legitimate users to break down the access control system of the smart home may vary from curiosity (e.g. a kid playing with oven setting), disturbing other family members (e.g. a kid locking his brothers outside the house), to disobedience (e.g. a kid is watching TV and he is not allowed to), or robbery (e.g. a worker trying to adjust the camera setting). Making sure that those legitimate users get access only to what they are authorized to by the house owner, is the central focus of our paper. We emphasize that authorized insiders who try to hack the access control system, or to break the IoT devices to assure that those legitimate users get access only to what they can access device k when the set of operation permissions when set of environment conditions is active iff the following predicate is true:

\[ (\exists (d_{n}, o_{k}) \in PDRA \land \exists (s_{i}, d_{n}) \in SR \land r_{p_{m}, ER} \subseteq \{er \in ER \mid (\exists EC'_{l} \subseteq EC_{l}) \} ) \]

VI. EGBRAC Model for Smart Home IoT

In this section we define the EGBRAC (Extended Generalized Role-Based Access Control) model.

A. EGBRAC Formal Definition

Fig. 2 depicts the components of EGBRAC, and Table II formally defines these. Sets are shown as ovals in Fig. 2, while the binary relations amongst them are shown as directed arrows with the single arrow indicating “one” and the double arrow “many.” An arrow ending in a dot indicates a subset rather a single element of that set (as in one end of EA and RPEA). A solid arrow represents assignment, a dashed arrow indicates an
association via mathematical definitions, for example, RPRA and RPEA relations are determined by definition from RP and hence are associations rather than independent assignments. A dotted arrow represents constraints.

Users (U), Roles (R), and Sessions (S) are familiar sets in RBAC systems. A user is a human being who interacts with smart home devices as authorized. In context of smart homes, a role specifically represents the relationship between the user and the family, which encompasses parents, kids, neighbors and such [2]. The many-to-many association via mathematical definitions, for example, RPRA can get access to all device roles in a combination of a role and currently active environment rp is a many to many association via mathematical definitions, for example, RPRA can get access to all device roles assigned to it through RPRA.

Evenings we can use D an operation on one device, i.e. it is a device, operation pair. Operations (OP) represent actions on devices as specified by device manufacturers. A permission is an approval to perform an operation on one device, i.e. it is a device, operation pair. The set of permissions P is a subset of \( D \times \Omega \). In EGRBAC, Device Roles (DR) are means of categorizing permissions of different devices (different from GRBAC where Device Roles categorize devices including all their permissions). For example, we can categorize the dangerous permissions of various smart devices by creating a device role called dangerous devices and assign dangerous permissions (such as, turning on the oven, turning on the mower, and opening and closing the front door lock) to it. The many-to-many DR is a role part that an active role has a role part \( r_p \), and an environment role part \( r_p.ER \) that is the subset of environment roles associated with \( r_p \). The permissible role pairs RP are specified as a subset of \( R \times 2^{ER} \), since some ER subsets may not be meaningful. RPRA associates each role to one or more role pairs. RPEA associates each role pair to a subset of ER. RPDRA brings all these components together by assigning device roles to role pairs, and hence, for each role pair \( r_p \), the single role associated to it through RPRA can get access to all device roles assigned to it through RPDRA, when the set of environment

\[
U = \{\text{alex, bob, susan, james, julia}\}
\]

\[
R = \{\text{kids, parents, babysitters, guests, neighbors}\}
\]

\[
UA = \{(\text{alex, kids}), (\text{bob, parents}), (\text{susan, babysitters}), (\text{james, guests}), (\text{julia, neighbors})\}
\]

\[
D = \{\text{TV, DVD, PlayStation, DoorLock, Oven}\}
\]

\[
OP = \{(\text{On, Off, PG, R, Lock, Unlock, On\_weekends, Off\_weekends})\}
\]

\[
P_1 = \{(\text{TV, DVD, PlayStation}) \times \{(\text{On, Off, PG, Lock, Unlock})\}\}
\]

\[
P_2 = \{(\text{DoorLock}) \times \{(\text{Lock, Unlock})\}\}
\]

\[
P_3 = \{(\text{Oven}) \times \{(\text{On\_weekends, Off\_weekends})\}\}
\]

\[
P = P_1 \cup P_2 \cup P_3
\]

\[
DR = \{\text{Dangerous\_Devices, Entertainment\_Devices, Kids\_Friendly\_Contents}\}
\]

**Fig. 3:** Use Case 1 Configuration in EGRBAC

roles which are associated to \( r_p \) through RPEA are active.

The main idea in EGRBAC as a whole is that a user is assigned a subset of roles and, according to the current active roles in a session and active environment roles, some role pairs will be active, whereby the user will get access to the permissions (not devices as in GRBAC) assigned to the device roles which are assigned to the current active role pairs.

The bottom part of Table II formalizes the authorization function of EGRBAC. Consider a session \( s_i \) which attempts to perform operation \( op_j \) on device \( d_j \) when the subset of environment conditions \( EC_l \) is active. This operation will succeed if and only if there is a role pair \( rp_m \) and device role \( dr_n \) assigned to each other in RPDRA such that the following conditions are true. (i) \( dr_n \) is assigned the permission \( (d_j, op_k) \) in PDRA. (ii) \( rp_m, r \) is one of the active roles of \( s_i \) (as given in \( SR \)). (iii) Each environment role \( er \in rp_m.ER \) is active because it is activated by a subset of the currently active environment conditions \( EC_l \).

**B. EGRBAC Use Case**

We present a use case to illustrate the components and configurations of EGRBAC. The objective is as follows. (a) Allow kids access to a subset of capabilities (On, Off, PG, but not R) in entertainment devices (TV, DVD, and PlayStation) during weekend evenings only. (b) Authorize parents to use dangerous capabilities of dangerous devices (i.e lock and unlock the door lock, switch on and off the oven) at any time. (c) Authorize parents, babysitter, guests, and neighbors to use entertainment devices any time unconditionally.

EGRBAC can be configured as shown in Fig. 3 to achieve
this objective. The five users alex, bob, susan, james, and julia, are respectively assigned to roles kids, parents, babysitters, guests and neighbors. The devices comprise DoorLock, Oven, TV, DVD and PlayStation. Each device has different permissions as indicated in P1, P3 and P4. Also P2 is a subset of P1, restricted to PG content.

We have three device roles with PDRA assigning P1 permissions to Entertainment_Devices, P2 to Kids_Friendly_Content, and P3 and P4 to Dangerous_Devices. Three environment conditions, weekends, evenings and TRUE are defined to be respectively active on weekends, evenings and always. EA specifies that the environment role Entertainment_Time is active when both environment conditions weekends and evenings are active while Any_Time is always active.

RPDRA has the following assignments. The role pair (parents, {Any_Time}) is assigned to the device role Dangerous_Devices, whereby parents can use permissions P3 and P4 without environmental restrictions. The role pair (kids, {Entertainment_Time}) is assigned to the device role Kids_Friendly_Content, so that kids are restricted to P2 permissions and only when the environment role Entertainment_Time is active. The role pairs (parents, {Any_Time}), (babysitters, {Any_Time}), (guests, {Any_Time}), (neighbors, {Any_Time}) are assigned to the device role Entertainment_Devices so that users with these roles can use all permissions on Entertainment_Devices at any time.

C. EGRBAC Constraints

An important component in EGRBAC is Constraints. A constraint is an invariant that must be maintained at all times. Constraints are an integral part of RBAC and ABAC models. In EGRBAC, we define three types of constraints, as follows.

Permission-role constraint. These constraints prevent specific roles from access to specific permissions. In the use case above, the permissions embodied in the Dangerous_Devices role are assigned to the (parents, {Any_Time}) role pair in RPDRA. However, this does not prevent assignment of Dangerous_Devices to other role pairs, perhaps even to (kids, {Any_Time}). The latter assignment could happen inadvertently or maliciously. Permission-role constraints prevent such situations.

Formally, PRConstraints \( \subseteq 2^P \times 2^R \) constitute a many to many subset of permissions to subset of roles relation. Each \( prc = (P_i, R_j) \in PRConstraints \) specifies the following invariant for every \( p_m \in P_i \) and every \( r_n \in R_j \):

\[
(\forall (r_{p_i}, d_{r_j}) \in RPDRA) \\
[(p_m, d_{r_j}) \notin PDRA \lor r_{p_i} \neq r_n]
\]

Thus, it is forbidden to assign any device role that \( p_m \) is assigned to, to any role pair with \( r_n \) as the role part. Use case 1 can be augmented with the constraint shown below.

\[ PRConstraints = \{(P_3 \cup P_4), R \setminus \{parents\}\} \]

This will prevent the assignment of any permissions in P3 or P4 to role pairs with the role part being any role except for parents.

Static Separation of Duty (SSD). This is the familiar SSD in RBAC. It enforces constraints on the assignment of users to roles. In other words, if a user is authorized as a member of one role, the user is prohibited from being a member of a second conflicting role [45].

Formally, SSDConstraints \( \subseteq R \times 2^R \) constitute a many to many role to a subset of mutually exclusive roles relation. Each \( ssdc = (r_i, R_j) \in SSDConstraints \) specifies the following invariant:

\[
(\forall u_m \in U)(\forall r_n \in R_j)\{(u_m, r_n) \in UA \Rightarrow (u_m, r_i) \notin UA\}
\]

Thus, it is forbidden to assign any role that is in \( R_j \) to any user to whom \( r_i \) is assigned.

Dynamic Separation of Duty (DSD). This is the familiar DSD in RBAC. With DSD it is permissible for a user to be authorized as a member of a set of roles which do not constitute a conflict of interest when acted in independently, but produce policy concerns when allowed to be acted simultaneously [45] in the same session.

Formally, DSDConstraints \( \subseteq R \times 2^R \) constitute a many to many role to a subset of active mutually exclusive roles relation. Each \( dsdc = (r_i, R_j) \in DSDConstraints \) specifies the following invariant:

\[
(\forall s \in S)(\forall r_n \in R_j)[(s, r_n) \in SR \Rightarrow (s, r_i) \notin SR]
\]

Thus, it is forbidden for any session that has role \( r_i \) active to also have any role \( r_n \in R_j \) active.

D. EGRBAC Assessment

We now show that EGRBAC meets our criteria for smart home IoT which proposed in Section II. Our model is dynamic since it can capture different environment conditions through environment roles. Moreover, through device roles EGRBAC enables users to give access to subsets of permissions of different devices instead of giving them access to the entire devices. Thereby, EGRBAC is a fine grained model. EGRBAC is suitable for constrained home environment, since it doesn’t require smart devices to implement a computational heavy logic. The enforcement architecture that we adopt (see Section IV) includes the component smart hub, which facilitates transferring the policy decision engine to a more capable local device. This enables devices to collect and analyze data externally, but closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. Moreover, mediating each request through smart hub solves the heterogeneity problem of IoT devices. EGRBAC is designed to fit smart home IoT access control challenges. EGRBAC is demonstrated with one illustrative use case, and an AWS implementation that captures local, and remote access for smart home devices as described in Section A, with accompanying performance analysis.

Limitations Except for relationships, our model doesn’t capture other user attributes. It does not handle device to device communication. Finally, it doesn’t consider continuous
verification for access control authorized policies, where the authorization predicate is only examined at the time of request. **Policy Conflicts** Conflicting policies may occur when you have negative policies, where you prevent specific roles from accessing specific permissions. In EGRBAC model our policies are positive policies where you give roles access to specific permissions. Instead of negative policies, EGRBAC uses constraints to prevent a specific role $r_m$ from accessing a specific permission $p_m$.

VII. PROOF-OF-CONCEPT IMPLEMENTATION

In this section we describe a proof-of-concept implementation of EGRBAC. We simulated the use case provided in Fig. 3 using AWS (Amazon Web Services) IoT service [46]. The simulation illustrates how the access control model and policies can be configured to establish the applicability of our model utilizing commercially available systems. Moreover, we executed multiple test cases to measure the processing time in different scenarios. The details of this section are provided in the appendix which can be accessed in the following url: https://profsandhu.com/conference_papers.htm

VIII. CONCLUSION AND FUTURE DIRECTIONS

In this paper, we propose EGRBAC access control model for smart home IoT. Our model fills the gap in the area of access control model for smart home IoT. It is a dynamic, fine-grained model that grants access based on the specific permission required rather than at device granularity. We demonstrated our model with a use case scenario and a proof-of-concept implementation in AWS. We also conducted a performance test to depict how our system responds in different scenarios with different loads, the results show that our model is functional, and applicable. Our model still needs some further work as discussed in Section VI. In the future we are planning, to develop a family (or series) of models ranging from relatively simple and complete to incorporating increasingly sophisticated and comprehensive features.

REFERENCES

APPENDIX

Here, we describe a proof-of-concept implementation of EGRBAC. We simulated the use case provided in Fig. 3 using AWS (Amazon Web Services) IoT service [46]. The simulation illustrates how the access control model and policies can be configured to establish the applicability of our model utilizing commercially available systems. Moreover, we executed multiple test cases to measure the processing time in different scenarios. The details of this section are provided in the appendix which can be accessed in the following url: https://profsandhu.com/conference_papers.htm

An AWS account is required to configure and deploy the AWS IoT service known as Greengrass. The Greengrass SDK (Software Development Kit) extends cloud capabilities to the edge, which in our case is the smart home. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely on local networks [47].

In our system Greengrass serves as a smart hub and a policy engine. It runs on a dedicated virtual machine with 1 virtual CPU and 2 GB of RAM running ubuntu server 16.04.5 LTS. Through AWS IoT management console, one virtual object (aka digital shadow) is created for each physical device and the two are cryptographically linked via digital certificates with attached authorization policies. Each simulated device is run on a separate virtual machine. These devices use MQTT protocol to communicate to the AWS IoT service with TLS security. Since the environment conditions in our use case are time based, they are directly sensed by Greengrass.

To enforce EGRBAC, we utilized two Json files UserRoleAssignment.json and policy.json, where UserRoleAssignment.json defines the assignments of users to their corresponding roles while policy.json defines all other EGRBAC components relevant to the use case. We also utilized the lambda function service in AWS IoT platform [48] to receive the operation requests of users to access the smart devices in the house, analyze each request according to the content of the policy.json and UserRoleAssignment.json files, and finally trigger the desired actions on the corresponding simulated devices. Code was in Python 2.7 and running on a long-lived lambda function with 128 MB Memory Limit, 30 second timeout. The lambda function, the UserRoleAssignment.json file, and the policy.json file are all configured in the Greengrass. We should mention that our system is a default deny system.

Fig. 4, illustrates how the communication is handled in our implementation when the user tries to send operation request to turn on a smart TV through his mobile phone while he is inside the house. In this case, a request is sent via MQTT protocol to the virtual object (or local shadow) corresponding to his phone in Greengrass. There is a publish/subscribe relation between the user’s phone, and the local shadow through the user’s private topic User/Shadow/Update. The user’s phone publishes to the topic User/Shadow/Update, and the local shadow gets notified with the request. After that, the local shadow publishes to the user’s private topic User/Status/Update, and then since the lambda function is subscribed to this topic it analyzes the request according to the policy.json and UserRoleAssignment.json files and makes a decision whether to allow the user to turn on the TV or not. At this point, there are two cases, either permission is granted or denied. If permission is denied, the lambda function publishes to the user’s public topic User/Status/Update, the local shadow gets notified and updates the user’s phone that the permission was denied. The smart TV in this case does not get an indication that a user attempted to access it. If permission is granted, the smart TV local shadow is notified through the device’s private topic Device/Shadow/Update and updates the smart TV with the turn on command. After the smart TV is turned on, it publishes to the device’s private topic Device/Status/Update and the TV local shadow is notified which further notifies the lambda function by publishing to the device’s public topic Device/Status/Update. The lambda...
TABLE III: One User Sending Requests to Multiple Devices

<table>
<thead>
<tr>
<th>Number of Users</th>
<th>Number of devices</th>
<th>Lambda Processing Time in ms</th>
<th>Total Number of requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.092318</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1.896767</td>
<td>5000 (1000 per request)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2.020956</td>
<td>5000 (1000 per request)</td>
</tr>
</tbody>
</table>

TABLE IV: Multiple Concurrent Instances of One User Sending Request to One Device.

<table>
<thead>
<tr>
<th>Number of Users</th>
<th>Number of devices</th>
<th>Lambda Processing Time in ms</th>
<th>Total Number of requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1.029138</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1.202856</td>
<td>5000 (1000 per request)</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1.236029</td>
<td>3000 (1000 per request)</td>
</tr>
</tbody>
</table>

TABLE V: Multiple Users Sending Requests to One Device

<table>
<thead>
<tr>
<th>Number of Users</th>
<th>Number of devices</th>
<th>Lambda Processing Time in ms</th>
<th>Total Number of requests</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.955532</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0.996821</td>
<td>5000 (1000 per request)</td>
</tr>
</tbody>
</table>

function then notifies the user’s phone’s local shadow which in turn updates the user’s phone that the TV was turned on successfully.

Fig. 5, illustrates how the communication is handled in our implementation in case of remote access. If a user Bob is trying to turn on the oven using his smart phone from a remote place. First, a request is sent through the HTTP send protocol to the cloud’s synchronized shadow state of the user device, in this case the user’s phone. Once the user’s phone state is changed on the cloud, the cloud forwards the message to the local Greengrass lambda by publishing to the user’s private topic User/Shadow/Update, the lambda receives the request, analyzes it according to the access control policies defined in the policy.json file and the UserRoleAssignment.json file and makes a decision to allow the user to turn on the oven or not. If the access is granted, the lambda function will send the request to the smart device Greengrass’s local shadow by publishing to the device’s private topic Device/Shadow/Update, the local shadow will get the request and will automatically update the smart device (smart oven in this case) to turn on. When the smart device perform the operation, it notifies its local shadow by publishing to the device’s private topic Device/Shadow/Update, the local shadow then notifies the lambda by publishing to the device’s public topic Device/Status/Update, the lambda then updates the user’s phone local shadow by publishing to the user’s public topic User/Status/Update. The user’s phone local shadow automatically synchronizes this state to the cloud shadow which in turn notifies the user’s phone that the request has been served. On the other hand, if the decision was not to allow this operation to be performed, the lambda function would publish to the user’s public topic User/Status/Update, the local shadow would get notified and would automatically synchronize this state to the cloud’s synchronized shadow state of the device. The cloud’s shadow would then update the user’s phone through the http send protocol that the permission was denied and the user has no right to turn on the oven. The smart oven, in this case, would never get an indication that a user attempted to access it.

A. Performance results

We executed multiple test cases to measure the processing time in different scenarios. In our performance testing, we implemented the configuration of Fig. 3. In the following test cases, we measure the average lambda function execution time under different conditions. Table III shows the average lambda function execution time when we send multiple requests from one user to multiple devices. The first, second, and third rows show the average time when the parent Bob requests to unlock the door lock, the average time when Bob requests to turn on the oven, the TV, and the DVD at the same time, and the average time when Bob requests to unlock the door lock, turn on the oven, the TV, the DVD, and the playStation at the same time respectively. All the requests were approved as they were supposed to according to our configured policies. Table IV shows the average lambda function execution time when we send multiple requests from multiple users to multiple devices (one user per device) at the same time. The first, second, and third row show the average time when the parent Bob requests to unlock the door lock, the average time when Bob requests to unlock the door lock, the kid Alex requests to turn on the oven, and the babysitter Susan requests to turn on the TV at the same time, the average time when the three access requests tested in the second row are carried again in addition to, the guest James requests to turn on the DVD, and the neighbor Julia requests to turn on the playStation. The system responded correctly where all the requests were approved except for when the kid Alex requests to turn on the oven. We can conclude that when the number of requests for different users and different devices (one user per device) increases, the lambda processing time also increase. Finally, Table V shows the average lambda function execution time when we send multiple requests from multiple users to one device at the same time. The first, second, and third rows show the average time when the parent (1 user), the parent, the kid, and the babysitter (3 users), or the parent, the kid, the babysitter, the guest, and the neighbor (5 users) respectively all request to unlock the door at the same time. The system responded correctly where all the requests were denied except for when the parent Bob requests to unlock the door lock. Here, we can see that the average of the lambda processing time decreases when we have more denies. This result is expected since in order to approve a request, our policy checking engine (the lambda function) implemented to check for the authorization predicate explained in Table II need to verify each condition in the authorization predicate. On the other hand, if only one of the authorization predicate conditions is violated the lambda function will deny the request without the need to check the rest of the authorization predicate. To conclude, our system takes more time when approving a request than when denying it. Overall, our model is functional, and can be easily applied. Moreover, we can notice that the execution time is generally low.