information

Article

An Attribute-Based Approach toward a Secured Smart-Home IoT Access Control and a Comparison with a Role-Based Approach

Safwa Ameer *, James Benson and Ravi Sandhu

Department of Computer Science, Institute for Cyber Security and CREST C-SPECC, The University of Texas at San Antonio, San Antonio, TX 78249, USA; James.Benson@utsa.edu (J.B.); ravi.sandhu@utsa.edu (R.S.)
* Correspondence: safwa.ameer@gmail.com

Abstract: The area of smart homes is one of the most popular for deploying smart connected devices. One of the most vulnerable aspects of smart homes is access control. Recent advances in IoT have led to several access control models being developed or adapted to IoT from other domains, with few specifically designed to meet the challenges of smart homes. Most of these models use role-based access control (RBAC) or attribute-based access control (ABAC) models. As of now, it is not clear what the advantages and disadvantages of ABAC over RBAC are in general, and in the context of smart-home IoT in particular. In this paper, we introduce $HABAC_\alpha$, an attribute-based access control model for smart-home IoT. We formally define $HABAC_\alpha$ and demonstrate its features through two use-case scenarios and a proof-of-concept implementation. Furthermore, we present an analysis of $HABAC_\alpha$ as compared to the previously published EGRBAC (extended generalized role-based access control) model for smart-home IoT by first describing approaches for constructing $HABAC_\alpha$ specification from EGRBAC and vice versa in order to compare the theoretical expressiveness power of these models, and second, analyzing $HABAC_\alpha$ and EGRBAC models against standard criteria for access control models. Our findings suggest that a hybrid model that combines both $HABAC_\alpha$ and EGRBAC capabilities may be the most suitable for smart-home IoT, and probably more generally.

Keywords: smart homes; IoT; access control

check for updates

Citation: Ameer, S.; Benson, J.; Sandhu, R. An Attribute-Based Approach toward a Secured Smart-Home IoT Access Control and a Comparison with a Role-Based Approach. *Information* **2022**, *13*, 60. https://doi.org/10.3390/info13020060

Academic Editor: Georgios Kambourakis

Received: 6 January 2022
Accepted: 21 January 2022
Published: 25 January 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction and Motivation

The Internet of Things (IoT) describes the network of physical objects (things) that are embedded with sensors, software, and other technologies to connect and exchange data with other devices and systems over the Internet [1]. IoT has been used in a wide variety of applications, including infrastructure applications (smart cities, energy management), consumer applications (smart homes, elder care), organizational applications (medical and health care, vehicular communication systems), industrial applications (manufacturing, agriculture), and military applications (Internet of Battlefield Things).

In 2017, the global market for the Internet of Things (IoT) surpassed USD 100 billion in revenue for the first time, and forecasts suggest that the figure will rise to USD 1.6 trillion by 2025. As a result of such a prognosis, it is predicted that the technology will reach heights that no one could imagine. However, with the growing popularity of IoT devices, there will be an increase in IoT security concerns.

Smart homes are among the most popular areas for deploying smart connected devices. They are changing our lifestyles. However, with new opportunities come new challenges. Surprisingly little attention has been paid to access control policy specification and authentication in home IoT. Smart homes differ from traditional computing domains in many notable ways [2]. Home-IoT users have complex social relationships and use the same devices. In addition, most smart-home devices lack screens and keyboards, which makes them more convenient, but makes access control management more difficult. Several real-world examples of the deficiencies of current policies and authentication procedures for controlling access to home-IoT devices have emerged [2–4].

To attain the best results and minimize risks and threats, an organization must select an appropriate access control model in light of its diverse structure, requirements, and specifications. Several access control models have been described in the literature for IoT in general, a few of which have been developed specifically for smart homes. Most of the proposed IoT AC models are based on ABAC or RBAC. According to some researchers, RBAC is better suited for IoT since it is easier to manage and review, whereas ABAC is more challenging [5–7]. Other researchers assert that ABAC models are more scalable and dynamic because they can capture information about different devices and contexts. However, RBAC models can be extended to be dynamic and fine-grained. For example, the recent EGRBAC (extended generalized role-based access control) model [8] for smart-home IoT. EGRBAC can express environment and device characteristics and is suitable for constrained home environments. Therefore, in the case of smart homes, it is not completely clear what the advantages of ABAC over RBAC are, and vice versa.

In this paper, we introduce $HABAC_\alpha$, an attribute-based access control model for smart-home IoT. Our model is dynamic and fine-grained. It captures users, environment, operations, and devices characteristics. We provide a detailed formal definition of our model and illustrate its features through two use-case scenarios and a proof-of-concept implementation.

Furthermore, we compare $HABAC_\alpha$ to the previously published EGRBAC model for smart-home IoT [8]. We chose to compare $HABAC_\alpha$ against EGRBAC for the following reason. EGRBAC is a dynamic contextual-aware RBAC-based access control model specifically designed to meet smart-home challenges. Our primitive insight is that a hybrid approach will better address smart-home IoT access control requirements, as this was the case for some traditional access control domains. However, this insight needs to be further explored by comparing RBAC and ABAC-based models specifically defined to meet smart-home challenges. In addition, this comparison will serve as a guide in developing appropriate hybrid models. Toward this goal, in this paper, we provide approaches for constructing $HABAC_\alpha$ specification from EGRBAC and vice versa to compare the theoretical expressiveness power of these models. Then, we evaluate these models against standard criteria for access control models adapted from [9]

The paper is organized as follows. In Section 2, we provide an analysis and review of related work, including an overview of EGRBAC [8]. In Section 3, we introduce the $HABAC_\alpha$ model along with two use-case scenarios and a proof-of-concept implementation. Furthermore, we conducted different test scenarios to depict the performance of our implementation. In Section 4 we compare between EGRBAC and $HABAC_\alpha$ in terms of theoretical expressiveness power. In Section 5 we conduct a comprehensive theoretical comparison between EGRBAC and $HABAC_\alpha$ against standard criteria for access control models. Section 6 discusses the insights of this work. Finally, Section 7 concludes the paper.

2. Related Work

Presently, IoT is one of the most researched topics in the literature. However, the entire IoT ecosystem still faces security challenges, from manufacturers to users. A number of researchers have studied IoT security and privacy issues [10–15]. Furthermore, some researchers have investigated the security risks and design concerns of IoT frameworks [15–20]. Most researchers generally accept that access control is a critical service in IoT.

The literature has presented many access control solutions (users to devices and/or devices to devices) that apply to different IoT applications. In [21], the authors extensively surveyed access control in IoT environments. However, few of them explicitly addressed smart-home challenges.

Some solutions rely on RBAC [22,23] (as in [5,8,24–29]). Other solutions, however, rely on ABAC [30,31]. For instance, the authors in [32] introduced an ABAC-based model that focuses on device-to-device access control. However, they did not illustrate their model through a performance analysis or at least a use-case scenario. In [33,34] the authors focused on providing sophisticated attribute-based encryption (ABE) models for smart grids, while they did not discuss the ABAC models that they consider. Additionally, for computationally constrained smart-home devices, an ABE model may not be appropriate. In [35] the authors introduced a formalized dynamic and fine-grained ABAC model for smart cars, which does not apply to the smart-home case. Recently, Bhatt et al. [36] proposed a conceptual attribute-based access control and communication control model for IoT. However, their access control model does not capture environment attributes. Our proposed $HABAC_\alpha$ model is an ABAC-based model that is dynamic and fine-grained. Moreover, it captures the characteristics of users, environments, operations, and devices.

Some researchers argue that contrary to ABAC-based models, RBAC-based models fail to capture changing characteristics such as environment attributes (time of the day, weather information) and device characteristics (device type, device state). However, the authorization process in RBAC is much simpler. Therefore, a combined access control model has been proposed by [6].

Additionally, the literature proposes several models that are based on blockchain technology [37–41]. However, as described by [39], blockchain technology possesses some technical characteristics that could limit its application, such as processing time and cryptocurrency fees. Only a few solutions based on UCON [42–44] have been presented in the literature, e.g., [45–47]. There have been some other access control models proposed for the Internet of Things. For example, in [48] the authors presented a certificate-based device access control scheme in an IoT environment. Researchers in [7,21,49–51] conducted surveys on different IoT access control models in the literature.

The authors of [2] presented a new perspective on home-IoT access control policies. They argued that smart-home IoT has unique characteristics that necessitate a rethinking of access control. However, few solutions in the literature are proposed specifically to meet smart-home IoT challenges. Here are some examples. In [32], the authors provided a device-to-device ABAC access control framework for smart homes. The authors in [24] introduced an RBAC-based access control model for aware homes. Moreover, in [52] the authors used identity-based encryption to implement a function-based access control model in smart homes. In [53], the authors proposed a protocol for authentication and key exchange in smart homes. Researchers in [54] developed a high-level access control mechanism for a multi-user and multi-device smart-home environment.

Based on analysis by He et al. [2] and a survey by Ouddah et al [21], Ameer et al [8] recently presented criteria for home-IoT access control models. Moreover, the same authors proposed the EGRBAC model. It is a policy model for smart-home IoT access control, which is based on RBAC and conforms to both of the [2,8] characteristics. As opposed to traditional RBAC, EGRBAC captures contextual environmental changes and different devices characteristics. Therefore, it refutes the argument that RBAC-based models are unsuitable and rigid when dealing with changes in the environment and device or permissions characteristics. Therefore, it is not entirely clear what the advantages of ABAC over RBAC are, and vice versa, in the setting of home IoT. In this paper, we analyze $HABAC_\alpha$ (our proposed ABAC-based model) compared to EGRBAC [8] (a dynamic, fine-grained RBAC-based model), and vice versa. We briefly review this model below since it is relevant to Sections 4.1 and 4.2.

2.1. Background: EGRBAC Model

Ameer et al introduced the extended generalized role-based access control model [8]. It is a dynamic, fine-grained RBAC-based model that grants access based on the specific permission required rather than at device granularity. In addition to the usual concept of user roles, EGRBAC incorporates the notion of device roles and environment roles. This is illustrated in Figure 1.

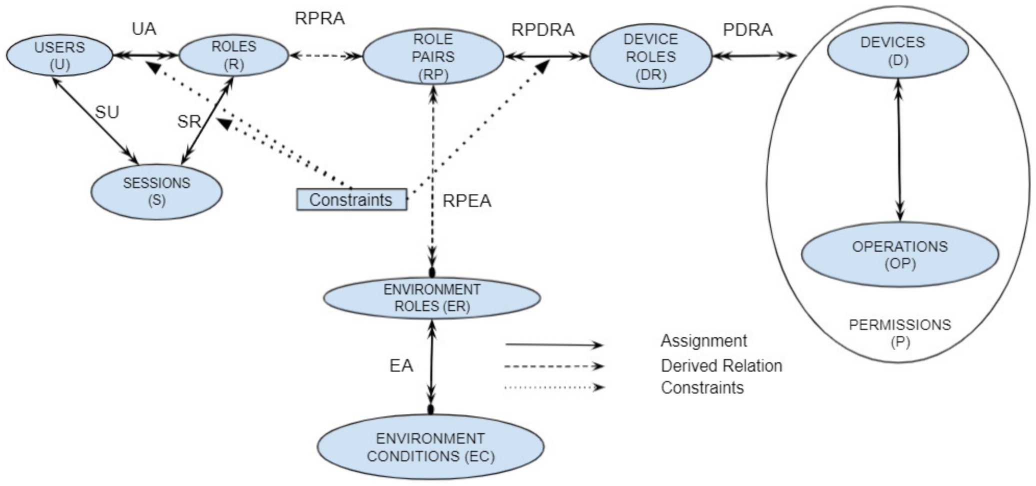

Figure 1. EGRBAC model components.

EGRBAC uses the familiar User (U), Role (R), and Session (S) sets in RBAC [22,23]. The term "user" refers to a person who uses smart-home devices as authorized. Roles (R) are similar to the traditional RBAC user roles. Nevertheless, in the case of smart homes, a role explicitly represents the relationship between a user and his or her family. Roles are assigned to different users through the many-to-many relationship UA. The system allows users to establish sessions to activate a subset of the roles they have been assigned to. A user may have more than one session active at the same time. SU is a many-to-one relationship that maps each session to its unique, controlling user. SR is a many-to-many relationship that maps each session to the set of roles associated with it.

A Device (D) is a smart-home device such as a smart door lock. Operations (OP) represent actions performed on devices according to manufacturer specifications. A permission is an approval to perform an operation on one device, i.e., it is a device–operation pair. Furthermore, as a way of categorizing permissions for different devices, device roles (DR) are defined. For example, we can categorize the dangerous permissions of various smart devices by creating a device role called dangerous devices and assigning 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 PDRA relationship specifies this assignment.

Environmental contexts, such as daytime/nighttime and winter/summer, are captured through the environment roles (ER) component. Environment roles are turned on/off (i.e., triggered) by environment conditions (EC). EA maps each environment role er_i to the subset of related environment conditions EC_i. It implies that when each $ec_i \in EC_i$ is active, the environment role er_i is also active. For example, suppose the environment role $Entertainment\_Time$ should be active on weekend evenings. To express this environment role in EGRBAC, we create the environment condition $weekends$, which will be active during the weekend, and the environment condition $evenings$, which will be active during the evening and add the relationship $(\{weekends, evenings\}, Entertainment\_Time)$ to the set EA.

Each role pair $rp_i \in RP$ combines a role and a subset of environment roles. A role pair rp has a role part $rp.r$ that is the single role associated with rp, and an environment role part $rp.ER$ denotes the set of environment roles associated with rp and is a subset of ER. RPRA associates each role to one or more role pairs. RPEA associates each role pair to a subset of ER. A role pair rp_i is active, when each environment role er_i in the set of environment role ER_i is active, where ER_i is associated with rp_i through the relation RPEA.

All these components are brought together by RPDRA, which assigns device roles to role pairs. Therefore, for each role pair rp, the single role associated with it through RPRA $rp.r$ can access all device roles assigned to it through RPDRA, as long as the set of environment roles associated with rp through RPEA, which is $rp.ER$ is active.

Essentially, the basic idea in EGRBAC is that a user is assigned a subset of roles and, according to the current active roles in a session and the current active environment roles, some role pairs will be active, giving the user access to the permissions assigned to the device roles assigned to the currently active role pairs. Figure 1 illustrates EGRBAC.

EGRBAC describes three types of constraints. (a) Permission–role constraints. These constraints prevent specific roles from accessing specific permissions during assignment time. This type of constraint is applied to the RPDRA relationship. For example, if we have a permission–role constraint prc_i, where $prc_i = (\{(Oven, On)\}, \{Kids\})$. This constraint prevents any $rpdra_i$ assignment relation from being added to the set RPDRA, if $rpdra_i$ gives the role $Kids$ access to the permission $(Oven, On)$. (b) Static Separation of Duty (SSD) is the familiar SSD in RBAC. It enforces constraints on the assignment of users to roles. This type of constraint is applied to the UA relationship. (c) Dynamic Separation of Duty (DSD) 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 that do not constitute a conflict of interest when acted independently but produce policy concerns when allowed to be acted simultaneously [55] in the same session. This type of constraint is applied to the SR relationship.

3. $HABAC_\alpha$ Model for Smart-Home IoT

Models based on ABAC are arguably well-suited for sophisticated domains, such as smart homes. By using attributes of users, sessions (subjects), environments, operations, and objects, they can specify dynamic, flexible, and fine-grained authorization policies. This section introduces our $HABAC_\alpha$ (Home-IoT Attribute-Based Access Control) model. This model governs smart-home user-to-device interactions.

3.1. Formal Definition

The $HABAC_\alpha$ model is inspired by [31,56]. The basic model components are illustrated in Figure 2 and are formally defined in Tables 1 and 2. It consists of four sets as follows: Users (U), Environment States (ES), Operations (OP), and Devices (D). These sets are presented in ovals. Moreover, attribute functions are presented in squares. We have four attribute functions, User/Session Attributes (USA), Environment-State Attributes (ESA), Operation Attributes (OPA), and Device Attributes (DA). The rectangles indicate two types of constraints: constraints on user attributes and constraints on session attributes.

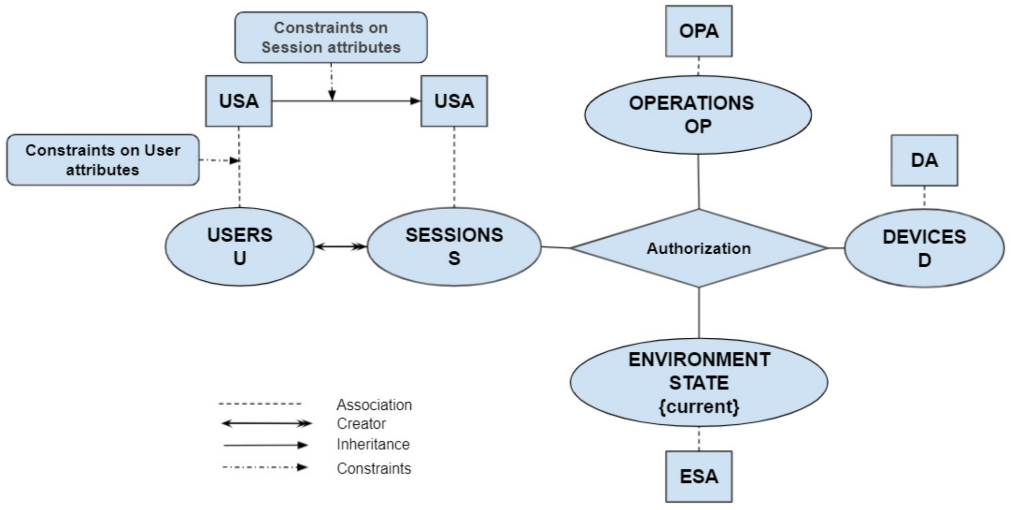

Figure 2. $HABAC_\alpha$ Model.

Table 1. $HABAC_\alpha$ Model formalization part I: basic components.

Basic Sets and Functions

– U is a finite set of users (homeowner-specified)
– S is the set of sessions (each session is created, terminated and controlled by an individual user)
– S is the set of sessions (each session is created, terminated and controlled by an individual user)
– The function $user(s) : S \to U$ maps each session to its unique creator and controlling user
– D is the set of devices deployed in the smart home (homeowner-deployed)
– OP is the set of possible operations on devices (device manufacturer-specified)
– The function $ops : D \to 2^{OP}$ specifies the valid operations for each device (device manufacturer-specified)
– $ES = \{current\}$ is a singleton set where *current* denotes the environment at the current time instance

Attribute functions and values

– USA, DA, OPA and ESA are sets of user/session, device, operation and environment-state attribute functions respectively, where for convenience we require USA, DA, OPA and ESA to be mutually exclusive
– Each session s inherits a subset of the attribute functions in USA from its unique user creator (controlled by the session creator $user(s)$). For every inherited attribute function $att \in theUSA$, $att(s) = att(user(s))$ at all times unless otherwise specified use of a non-inherited session attribute in a logical formula renders that formula false
– For each attribute att in $USA \cup DA \cup OPA \cup ESA$, $Range(att)$ is the attribute range, a finite set of atomic values
– $attType : USA \cup DA \cup OPA \cup ESA \to \{set, atomic\}$.
– Each $att \in theUSA \cup DA \cup OPA \cup ESA$ correspondingly maps users in U / sessions in S, devices in D, operations in OP or the environment-state *current* to atomic or set attribute values. Formally:

$$att : U \text{ or } S \text{ or } D \text{ or } OP \text{ or } \{current\} \to \begin{cases} Range(att), & \text{if } attType(att) = atomic \\ 2^{Range(att)}, & \text{if } attType(att) = set \end{cases}$$

– Every $att \in theUSA \cup DA \cup OPA \cup ESA$, att is designated to be either a static or dynamic attribute where dynamic attributes must have corresponding sensors deployed in the smart home (under homeowner control)
– Static attribute ranges and values are set and changed by administrator actions (by homeowner or device manufacturer)
– Dynamic attribute ranges and values automatically determined by sensors deployed in the smart home (under homeowner control)

Constraints

– $UAConstraint \subseteq UAP \times 2^{UAP}$ is the user attribute constraints relationship (homeowner-specified) where

$$UAP = \{(usa, v) \mid usa \in theUSA \ \wedge \ v \in Range(usa))\}$$

Each $uac = ((usa_x, v_y), UAP_j) \in UAConstraint$ specifies the following invariant:

$$\begin{cases} (\forall u_l \in U)(\forall (usa_m, v_n) \in UAP_j)[usa_x(u_l) = v_y \Rightarrow usa_m(u_l) \neq v_n], & \text{if } attType(usa_x) = attType(usa_m) = atomic \\ (\forall u_l \in U)(\forall (usa_m, v_n) \in UAP_j)[v_y \in usa_x(u_l) \Rightarrow v_n \notin usa_m(u_l)], & \text{if } attType(usa_x) = attType(usa_m) = set \end{cases}$$

Table 1. *Cont.*

– $SAConstraint \subseteq UAP \times 2^{UAP}$ is the session attribute constraints relationship (homeowner-specified) Each $sac = ((usa_x, v_y), UAP_j) \in SAConstraint$ specifies the following invariant:

$$\begin{cases} (\forall s_l \in S)(\forall (usa_m, v_n) \in UAP_j)[s_l \text{ inherits } usa_x \wedge usa_x(user(s_l)) = v_y \wedge usa_m(user(s_l)) = v_n \Rightarrow s_l \text{ does not inherit } usa_m], \\ \qquad \text{if } attType(usa_x) = attType(usa_m) = atomic \\ (\forall s_l \in S)(\forall (usa_m, v_n) \in UAP_j)[s_l \text{ inherits } usa_x \wedge v_y \in usa_x(user(s_l)) \wedge v_n \in usa_m(user(s_l)) \Rightarrow s_l \text{ does not inherit } usa_m], \\ \qquad \text{if } attType(usa_x) = attType(usa_m) = set \end{cases}$$

Attributes Authorization Function
– $Authorization(s : S, op : OP, d : D, current : ES)$ is a logic formula defined using the grammar of Table 2 (homeowner-specified). It is evaluated for a specific session s_i, operation op_k, device d_j and environment-state $current$ as specified in Table 2

CheckAccess Predicate
– $CheckAccess$ is evaluated when session s_i attempts operation op_k on device d_j in context of environment-state $current$
– $CheckAccess(s_i, op_k, d_j, current)$ evaluates to true or false using the following formula: $op_k \in ops(d_j) \;\wedge\; Authorization(s_i, op_k, d_j, current))$

Table 2. $HABAC_\alpha$ Model formalization part II: attribute authorization function.

Attribute authorization function
– $Authorization(s : S, op : OP, d : D, current : ES)$ is a first-order logic formula specified using the following grammar.

- $\alpha ::= term \mid term \wedge term \mid term \vee term \mid (term) \mid \neg term \mid \exists x \in set.\alpha \mid \forall x \in set.\alpha$
- $term ::= set\ setcompare\ set \mid atomic \in set \mid atomic \notin set \mid atomic\ atomiccompare\ atomic$
- $setcompare ::= \subset \mid \subseteq \mid \not\subseteq$
- $atomiccompare ::= < \mid = \mid \leq$
- $set ::= usa(s) \mid opa(op) \mid esa(current) \mid da(d)$, where $attType(usa) = attType(opa) = attType(esa) = attType(da) = set$
- $atomic ::= usa(s) \mid opa(op) \mid esa(current) \mid da(d) \mid value$, where $attType(usa) = attType(opa) = attType(esa) = attType(da) = atomic$

– For a specific session s_i, device d_j and operation op_k, the authorization function $Authorization(s_i, op_k, d_j, current)$ is evaluated by substituting the actual attribute values of $usa(s_i)$, $da(d_j)$, $opa(op_k)$ and $esa(current)$ for the corresponding symbolic placeholders and evaluating the resulting logical formula to be true or false. Any term that references an undefined attribute value is evaluated as false

3.1.1. Basic Sets and Functions

User set (U) refers to human beings communicating directly with smart objects (things). The system allows users to create sessions during which they can perform some operations on the system. Sessions (S) operate similarly to subjects in [31]. Only the user who initiates the session has the authority to terminate it. Each session s is associated with its unique user creator by the function $users(s)$.

Devices (D) refer to smart-home devices (smart things); for instance, smart TV. Operations (OP) are activities performed on devices in accordance with manufacturer specifications. The function $ops(d)$ maps each device d to the set of valid operations on d.

The Environment States set ($ES = \{current\}$) is a singleton set that only includes the state "current". The state $current$ denotes the picture of the environment at the current time instant. However, future extensions of this component, for instance, may be $ES = \{current, yesterday, lastweek\}$, which include other instants of time such as yesterday and last week. However, such generalization requires a careful investigation on how we will track time and to what extent.

3.1.2. Attribute Functions and Values

Each of the users, sessions, environment states, devices, and operations possess characteristics that are expressed as their attributes. An attribute is a function that takes an element such as a user and returns a certain value from its range. User/Session attribute functions set (USA) is the set of attributes associated with both users and sessions. Each session s inherits a subset of the attributes of its unique user creator. This is controlled by the unique user creator $user(s)$. If a session s inherited a user session attribute usa from his user $user(s)$, then it is required that $usa(s) = usa(user(s))$. The device attribute functions set (DA) consists of attributes related to smart devices, such as "kitchen devices" and "Alex devices". The operation attribute functions set (OPA) is a collection of attributes corresponding to different operations. For example, if you wish to characterize kid-friendly operations, you may create an operation attribute entitled "Kid-Friendly Operations" and associate it with those operations. Environment-state attribute functions set (ESA) describes the environment condition of the current instance of time. For example, "day", "time", and "weather condition".

To better demonstrate the concept of attributes, we will use the use case illustrated in Figure 3. We should mention that this use case is incomplete since it does not contain an authorization function. However, the main purpose is to illustrate the concept of attributes. In this use case, we have three users bob, $alex$, and $anne$. The user/session attribute function $Relationship$ captures the user relationship to the home. From Figure 3 we can notice that bob is the parent, $alex$ is the kid, and $anne$ is the teenager in the house. The user/session attribute function $UserLocation$ captures the user's current location inside the house. Moreover, we have four devices TV, $Oven$, $Fridge$, and $FrontDoor$. We have one device attribute function $DangerouseKitchenDevices$. $DangerouseKitchenDevices$ captures whether the device is a dangerous kitchen device or not. Finally, we have two environment-state attribute functions, day and $UsersInTheHouse$, which capture the current day and users inside the house, respectively.

The main reason for using different sets of attribute functions for different components in our model is that different components have different attribute functions. For instance, the user/session attribute function $Relationship$ is not relevant to the set of devices D. Moreover, the device attribute function $DangerouseKitchenDevices$ does not apply to the set of users. We cannot consider a specific user as a dangerous kitchen device; the evaluation $DangerouseKitchenDevices(bob)$ is not semantically correct.

Basically, an attribute range is composed of a finite set of atomic values. Each attribute function att_i has a range $Range(att_i)$, where $Range(att_i)$ represents the range of values to which the attribute function att_i can be evaluated. For example, in Figure 3, the user/session attribute function $Relationship$ range is $\{parent, kid, teenager\}$, the user/session attribute function $UserLocation$ range is $\{Bedroom1, Bedroom2, Gameroom, Kitchen, Livingroom, Bathroom1,$

$Bathroom2\}$. Similarly, the device attribute function $DangerouseKitchenDevices$ range is $\{True, False\}$, the environment-state attribute function day range is $\{S, M, T, W, Th, F, Sa\}$, and finally the environment-state attribute function $UsersInTheHouse$ range is the set of users U.

Each attribute function is an atomic-valued attribute or a set-valued attribute. An atomic-valued attribute will return exactly one value from its range. In the use case given in Figure 3, the user/session attribute functions $Relationship$ and $UserLocation$ are both atomic-valued attributes since they map different users into one value only from the attribute range. The device attribute function $DangerouseKitchenDevices$ is an atomic-valued attribute too. It takes one device as an input and returns only one value from the attribute function range as an output, either $True$ or $False$, since one device cannot be dangerous and non-dangerous at the same time. Similarly, the environment-state attribute function day is an example of an atomic-valued attribute.

On the other hand, a set-valued attribute will return a subset of the range, and not only one value from the range as in the case of atomic-valued attribute functions. For instance, in the use case given in Figure 3, the environment-state attribute function $UsersInTheHouse$ is a set-valued attribute. $UsersInTheHouse$ attribute function has a range equal to the set of users U, and it maps the environment state to a subset of values from that range. It returns a set with the names of users who are currently inside the house.

Moreover, we distinguish between two types of attributes: static and dynamic. All attributes indeed may change over a long time. However, some attributes are "relatively" static, as they tend to remain static (they evaluate to the same values) over a long period of time. Setting and changing the values of static attributes may require administrator intervention. For example, in our use case, the user/session attribute function $Relationship$ is considered static. To further illustrate this, let us consider the user $alex$. The attribute function $Relationship$, in this case, evaluates to kid, which tends to remain constant for a long time till $alex$ grows up and becomes a teenager, at which point an administrator action is required to change the value of this attribute to $teenager$; as a result, $Relationship(alex)$ will evaluate to $teenager$. Similarly, $DangerouseKitchenDevices$ is considered a static device attribute too.

On the other hand, dynamic attributes are always changing due to various circumstances, such as time of the day, user location, etc. In our use case, $UserLocation$, day, and $UsersInTheHouse$ are all considered to be dynamic attributes. Values of dynamic attributes are automatically determined by sensors deployed in the smart home and under homeowner control.

User/session attribute functions, device attribute functions, operation attribute functions, and environment-state attribute functions can all be classified as static or dynamic attribute functions. Moreover, all attribute functions can be classified as static or dynamic, whether atomic-valued or set-valued. However, the differentiation between static and dynamic attribute functions is unnecessary for how the model works formally. The process for triggering different attributes is outside the scope of our model.

Operation and device attribute functions are partial functions. Hence, some devices or operations will not be associated with some attributes. User/Session and environment-state attributes, on the other hand, are total functions.

In mathematics, a partial function f from a set X to a set Y is a function from a subset S of X (possibly X itself) to Y [57]. Since operation attribute functions and device attribute functions are defined as partial functions, in the use case illustrated in Figure 3, the device attribute function $DangerouseKitchenDevices$ does not have to map each device d in the set of devices D into a value from the range $\{True, False\}$. The $DangerouseKitchenDevices$ attribute function only maps the devices $Oven$ and $Fridge$ into the values $True$ and $False$ respectively. However, it does not map the devices TV and $FrontDoor$ into any value. In other words, $DangerouseKitchenDevices(TV)$ will be evaluated to be undefined.

On the other hand, total functions are defined for all elements in its domain. That being said, any static or dynamic attribute function att in the set of user/session attribute functions USA is defined for every user u_i in the set of users U. For instance, the user/session attribute function $Relationship$ is defined for every user u in the set of users U. Moreover, any static or dynamic environment-state attribute function att in the set of environment-state attribute functions ESA is defined for every environment state es in the set of environment state ES.

$U = \{bob, alex, anne\}$,
$USA = \{Relationship, UserLocation\}$
$Relationship : u : U \rightarrow \{parent, kid, teenager\}$
$Relationship : s : S \rightarrow \{parent, kid, teenager\}$
$UserLocation : u : U \rightarrow \{Bedroom1, Bedroom2, Gameroom, Kitchen, Livingroom, Bathroom1, Bathroom2\}$
$UserLocation : s : S \rightarrow \{Bedroom1, Bedroom2, Gameroom, Kitchen, Livingroom, Bathroom1, Bathroom2\}$
$Relationship(bob) = parent$
$Relationship(alex) = kid$
$Relationship(anne) = teenager$

$D = \{TV, Oven, Fridge, FrontDoor\}$
$DA = \{DangerouseKitchenDevices\}$
$DangerouseKitchenDevices : d : D \rightarrow \{True, False\}$
$DangerouseKitchenDevices(Oven) = True$
$DangerouseKitchenDevices(Fridge) = False$
All other values are undefined

$ES = \{Current\}$
$ESA = \{day, UsersInTheHouse\}$
$day : es : ES \rightarrow \{S, M, T, W, Th, F, Sa\}$
$UsersInTheHouse : es : ES \rightarrow 2^U$

Figure 3. Attributes Use Case.

3.1.3. Constraints

These are invariants that must never be violated. There are two types of constraints defined by $HABAC_\alpha$. First is constraints on user attributes. These constraints impose restrictions on user attributes. In other words, if a specific attribute value is assigned to a user, the user is prohibited from being assigned to another attribute value. For example, consider the following user attribute constraint:

$$uac_i = ((Relationship, kid), \{(Adults, True)\})$$

The above constraint implies that for any user u if $Relationship(u) = kid$, then it is required that $Adults(u) \neq True$.

The second type of constraint is constraints on session attributes. These constraints restrict the attributes that may be used in sessions. Here, it is permissible for a user to be assigned to different attribute values that do not constitute a conflict of interest when inherited independently by different sessions but produce policy concerns when allowed to be inherited together in the same session.

The concept of constraints is an integral part of $HABA_\alpha$, and it is a powerful mechanism for laying out higher-level policy. When specific attribute values are declared to be mutually exclusive, there is less concern about assigning conflicting or mutually exclusive attribute values to individual users. $HABAC_\alpha$ is an operational access control model. Managing and enforcing constraints is part of the administrative access control, which is outside the scope of this manuscript.

3.1.4. Attributes Authorization Function

A two-valued Boolean function is evaluated for each access decision. It is defined using the grammar of Table 2. For a specific session s_i, operation op_k, and device d_j the authorization function $Authorization(s_i, op_k, d_j, current)$ is evaluated by substituting the actual attribute values of $usa(s_i)$, $da(d_j)$, $opa(op_k)$ and $esa(current)$ for the corresponding symbolic placeholders and evaluating the resulting logical formula to be true or false. Any term that references an undefined attribute value is evaluated as false.

The term refers to any atomic logical declarative sentence. An atomic sentence is a type of declarative sentence that is either true or false and which cannot be broken down into other simpler sentences [58]. Consider Use Case A for more illustration.

$HABAC_\alpha$ is an operational access control model. Authorization function creation, check, and management tasks are considered part of administrative access control, hence outside the scope of this model.

3.1.5. Check Access Predicate

The $CheckAccess$ predicate is evaluated in each access request. When a session s_i attempts operation op_k on device d_j in the context of environment-state $current$ the $CheckAccess$ $(s_i, op_k, d_j, current)$ predicate evaluates to be true if the following conditions are satisfied:

1. The operation op_k is assigned to the device d_j by the device manufacturer.
2. The authorization function is evaluated to be true.

3.2. Use Cases

In this section, we describe two use cases to illustrate the components and configurations of $HABAC_\alpha$.

3.2.1. Use Case A

In this use case, the goals are as follows: (a) Kids should be allowed to use kid-friendly operations on entertainment devices (G and PG contents on TV, $A3$ (games for group aged below three years old), and $A7$ (games for children under the age of seven) on PlayStation) during a specific time (weekend afternoons and evenings, and weekday evenings). (b) Teenagers should only be permitted to use dangerous kitchen devices (Oven) if one of their parents is present in the kitchen. (c) Provide teenagers with unconditional permission to use non-dangerous kitchen devices (Fridge). (d) Provide teenagers with unconditional access to entertainment devices. (e) Parents should be permitted to use any operation on any device without restrictions.

Figure 4 illustrates how $HABAC_\alpha$ can be configured to achieve these objectives. Here, we have five users, bob, $alex$, $suzanne$, $john$, and $anne$ with user attribute $Relationship$ respectively assigned to the values $parents$, $kids$, $kids$, $teenagers$, and $teenagers$. Upon creation of a session s, this session will automatically inherit a subset of attributes from $user(s)$. There are five devices TV, $PlayStation$, $Oven$, $Fridge$ and $FrontDoor$. $Oven$, and $Fridge$ are assigned the following device attributes by the house owner $Fridge \leftarrow (DangerouseKitchenDevices$: $False)$, and $Oven \leftarrow (DangerouseKitchenDevices$: $True)$. There are 12 operations G, PG, $A3$, $A7$, $A12$, $BuyGames$, ON, OFF, $Open$, $Close$, $Lock$, and $Unlock$. These operations are assigned to operation attributes as follows: $(G$, $A3$, and $A7) \leftarrow (KidsFriendly$: $True)$, while $(PG$, $A12$, $BuyGames) \leftarrow (KidsFriendly : False)$. Since device attribute functions and operation attribute functions are partial functions, all other operations and devices attributes values are undefined. Any term that references an undefined attribute value is evaluated as false. The environment-state $current$ has three attribute functions (day, $time$, and $ParentInKitchen$).

The authorization function is a disjunction of five propositional statements. The disjunctions between different propositional statements are marked red for semantic illustration purposes only. In the first statement, children have access to kid-friendly operations on weekday evenings or weekends in the afternoons and evenings. Here, we have six terms (atomic declarative sentences) which are: (a) $Relationship(s) = kid$, (b) $day(current) \in \{Sa, S\}$, (c) 12:00 $\leq time(current) \leq$ 19:00, (d) $day(current) \in \{M, T, W, Th, F\}$, (e) 17:00 $\leq time(current) \leq$ 19:00, (f) $KidsFriendly(op) = True$. In the second propositional statement, teenagers are only permitted access to dangerous kitchen devices when one of their parents is present in the kitchen. Here, we have three terms, which are: (a) $Relationship(s) = teenager$, (b) $ParentInKitchen(current) = True$, (c) $DangerouseKitchen\ Devices(d) = True$. The third statement permits teenagers to use non-dangerous kitchen devices unconditionally. It contains two terms, which are: (a) $Relationship(s) = teenager$ (b) $DangerouseKitchen\ Devices(d) = False$. According to the fourth statement, teenagers are allowed to access both kid-friendly and non-kid-friendly operations without restriction. It contains three terms, which are: (a) $Relationship(s) = teenager$, (b) $KidsFriendly(op) = True$, (c) $KidsFriendly(op) = False$. Finally, the fifth statement guarantees parents access to anything at any time and contains one term: (a) $Relationship(s) = parent$.

$U = \{bob, alex, suzanne, anne, john\}$,
$USA = \{Relationship\}$
$Relationship : u : U \rightarrow \{parent, kid, teenager\}$
$Relationship : s : S \rightarrow \{parent, kid, teenager\}$
$Relationship(alex) = Relationship(suzanne) = kid$
$Relationship(anne) = Relationship(john) = teenager$
$Relationship(bob) = parent$

$D = \{TV, PlayStation, Oven, Fridge, FrontDoor\}$
$DA = \{DangerouseKitchenDevices\}$
$DangerouseKitchenDevices : d : D \rightarrow \{True, False\}$
$DangerouseKitchenDevices(Oven) = True$
$DangerouseKitchenDevices(Fridge) = False$
All other values are undefined

$ES = \{Current\}$
$ESA = \{day, time, ParentInKitchen\}$
$day : es : ES \rightarrow \{S, M, T, W, Th, F, Sa\}$
$time : es : ES \rightarrow \{x | x \text{ is an hour of a day }\}$
$ParentInKitchen : es : ES \rightarrow \{True, False\}$

$OP_{TV} = \{G, PG\}$
$OP_{PlayStation} = \{A3, A7, A12, BuyGames\}$
$OP_{Oven} = \{ON, OFF\}$
$OP_{Fridge} = \{Open, Close\}$
$OP_{FrontDoor} = \{Lock, Unlock\}$
$OP = OP_{TV} \cup OP_{PlayStation} \cup OP_{Oven} \cup OP_{Fridge} \cup OP_{FrontDoor}$
$OPA = \{KidsFriendly\}$
$KidsFriendly : op : OP \rightarrow \{True, False\}$
$KidsFriendly(G) = KidsFriendly(A3) = KidsFriendly(A7) = True$
$KidsFriendly(PG) = KidsFriendly(A12) = KidsFriendly(BuyGames) = False$
All other values are undefined

$Authorization(s : S, op : OP, d : D, current : ES) \equiv$
$(Relationship(s) = kid \wedge ((day(current) \in \{Sa, S\} \wedge 12{:}00 \leq time(current) \leq 19{:}00\ ) \vee$
$(\ day(current) \in \{M, T, W, Th, F\} \wedge 17{:}00 \leq time(current) \leq 19{:}00)\ ) \wedge$
$KidsFriendly(op) = True) \vee$
$(Relationship(s) = teenager \wedge ParentInKitchen(current) = True \wedge$
$DangerouseKitchenDevices(d) = True) \vee$
$(Relationship(s) = teenager \wedge DangerouseKitchenDevices(d) = False) \vee$
$(Relationship(s) = teenager \wedge (KidsFriendly(op) = True \vee KidsFriendly(op) = False)) \vee$
$(Relationship(s) = parent)$

Figure 4. Use Case A configuration.

3.2.2. Use Case B

The objectives of this use case are as follows: (a) Kids should be allowed to use kid-friendly operations on the iPad ($A5$, and $A8$ contents (applications for the group aged below five years old and below eight years old, respectively) during a specific time (weekends afternoons and evenings, and weekdays evenings). (b) If a parent is not present in the house, a teenager should not use dangerous devices (FrontDoor). (c) Give teenagers unconditional access to the iPad. (d) Parents should be permitted to use any operation on any device without restrictions.

Figure 5 illustrates how $HABAC_\alpha$ can be configured to achieve these objectives. Here, we have three users, bob, $suzanne$, and $john$ with user attribute $Relationship$ respectively assigned to the values $parents$, $kids$, and $teenagers$. Upon creation of a session s, this session will automatically inherit a subset of attributes from $user(s)$. There are three devices $iPad$, $FrontDoor$ and $lawnMower$. $FrontDoor$ is assigned to the following device attributes by the house owner $FrontDoor \leftarrow (DangerouseDevices : True)$. There are nine operations $A5$, $A8$, $A11$, $Games$, $Movies$, ON, OFF, $Lock$, and $Unlock$. These operations are assigned to operation attributes as follows: $(A5$ and $A8) \leftarrow (KidsFriendly : True)$, while $(A11, Games,$ and $Movies) \leftarrow (KidsFriendly : False)$. Since device attribute functions and operation attribute functions are partial functions, all other operations and devices attributes values are undefined. Any term that references an undefined attribute value is evaluated as false. The environment-state $current$ has three attribute functions $(day, time,$ and $ParentInTheHouse)$.

The authorization function is a disjunction of four propositional statements configured to maintain the four objectives of this use case.

3.3. *Proof-of-Concept Implementation*

3.3.1. Enforcement Architecture

$HABAC_\alpha$ is an abstract policy model that can be used with different enforcement models. Each enforcement model can use different implementation models which incorporate different underlying technologies.

Here, we adopted the smart-home IoT enforcement architecture shown in Figure 6, which was first introduced by Geneiatakis et al. [59]. According to this architecture, IoT devices are directly connected to a corresponding hub. Thus, other devices and users cannot access them directly. Here, there are two types of access. Local access provides users with direct access to the IoT devices through the connectivity services provided by the hub. Remote access allows users to access IoT devices remotely via cloud services. These cloud services then communicate with the smart hub over the Internet.

Moreover, we used the Amazon Web Services (AWS) IoT service [60] as an implementation technology to implement Use Case A presented in Section 3.2. However, in our enforcement, we only handled local access.

First, we created an AWS account, then we configured and deployed Greengrass [61]. Through Greengrass SDK (Software Development Kit), cloud capabilities can be extended to the edge, which is the smart home. Hence, Greengrass acts as a hub and a policy engine within the smart home. Greengrass was installed on a dedicated virtual machine with one virtual CPU, two GB of RAM, and running on Ubuntu server 18.04 LTS. Second, we simulated the five users (the devices which users use to access the smart things, e.g., their phones) and the five devices (the smart things that users want to access) of Use Case A using the AWS IoT device SDK for Python [62] provided by AWS on different virtual machines. Each machine has one virtual CPU and two GB of RAM running Ubuntu server 18.04.5 LTS. Third, we created one virtual object (digital shadow) for each physical device (smart thing devices needing access or a user access device) using the AWS IoT management console. Each physical device is cryptographically linked to its shadow by a digital certificate accompanied by authorization policies. Devices and users communicate with the AWS IoT service using MQTT protocol [63] with TLS security [64]. The MQTT standard is a lightweight machine-to-machine (M2M) publish/subscribe messaging protocol designed explicitly for use by constrained devices. Each shadow has a set of predefined MQTT topics/channels to interact with other IoT devices and applications.

$U = \{bob, suzanne, john\},$
$USA = \{Relationship\}$
$Relationship : u : U \to \{parent, kid, teenager\}$
$Relationship : s : S \to \{parent, kid, teenager\}$
$Relationship(bob) = parent$
$Relationship(suzanne) = kid$
$Relationship(john) = teenager$

$D = \{iPad, lawnMower, FrontDoor\}$
$DA = \{DangerouseDevices\}$
$DangerouseDevices : d : D \to \{True, False\}$
$DangerouseDevices(FrontDoor) = True$
All other values are undefined

$ES = \{Current\}$
$ESA = \{day, time, ParentInTheHouse\}$
$day : es : ES \to \{S, M, T, W, Th, F, Sa\}$
$time : es : ES \to \{x | x \text{ is an hour of a day }\}$
$ParentInTheHouse : es : ES \to \{True, False\}$

$OP_{iPad} = \{A5, A8, A11, Games, Movies\}$
$OP_{lawnMower} = \{ON, OFF\}$
$OP_{FrontDoor} = \{Lock, Unlock\}$
$OP = OP_{iPad} \cup OP_{lawnMower} \cup OP_{FrontDoor}$
$OPA = \{KidsFriendly\}$
$KidsFriendly : op : OP \to \{True, False\}$
$KidsFriendly(A5) = KidsFriendly(A8) = True$
$KidsFriendly(A11) = KidsFriendly(Games) = KidsFriendly(Movies) = False$
All other values are undefined

$Authorization(s : S, op : OP, d : D, current : ES) \equiv$
$(Relationship(s) = kid \wedge ((day(current) \in \{Sa, S\} \wedge 12{:}00 \le time(current) \le 19{:}00\ ) \vee$
$(\ day(current) \in \{M, T, W, Th, F\} \wedge 17{:}00 \le time(current) \le 19{:}00)\ ) \wedge$
$KidsFriendly(op) = True) \vee$
$(Relationship(s) = teenager \wedge ParentInTheHouse(current) = True \wedge$
$DangerouseDevices(d) = True) \vee$
$(Relationship(s) = teenager \wedge (KidsFriendly(op) = True \vee KidsFriendly(op) = False)) \vee$
$(Relationship(s) = parent)$

Figure 5. Use Case B configuration.

3.3.2. Use Case A Enforcement

To enforce Use Case A, we created four json files as follows, UsersAttributes.json, DevicesAttributes.json, OperationAttributes.json, and EnvironmnetAttributes.json to capture different users/session attributes, devices attributes, operations attributes, and environment attributes, respectively. How to automatically update different attributes values in these files is outside the scope of this work. Moreover, we used the AWS IoT lambda function to receive different user requests, analyze those requests according to the contents of the json files, allow or deny the requested accesses, and then trigger the corresponding devices to carry out the desired action. The code is written in Python 3.7 and runs on a long-lived lambda function with a memory limit of 500 MB and a timeout of 30 s.

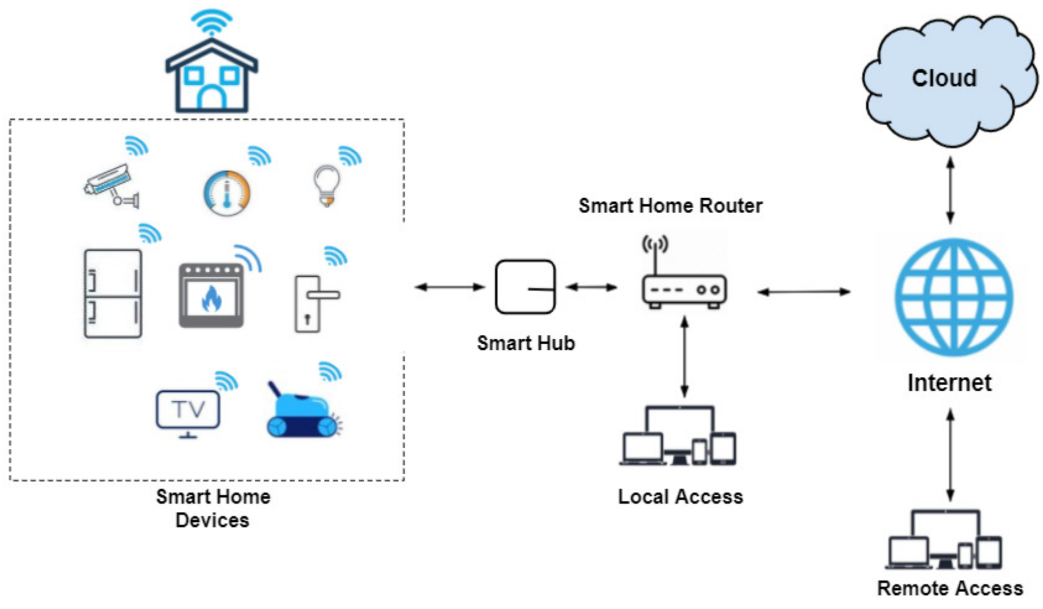

Figure 6. Enforcement architecture [59].

3.3.3. Local Communication Handling

Figure 7 depicts the sequence of actions in our local communication implementation. Sequence (a) illustrated in red demonstrates the sequence of actions when a request is denied. Sequence (b) in green illustrates the sequence of actions when a request is authorized. For instance, if the user wishes to turn on the smart oven using his smartphone from inside the house, first, through the publish/subscribe relationship between the user's phone and the local shadow, a request is initially sent via MQTT protocol to the virtual object (or local shadow) corresponding to the user phone in Greengrass. Upon receiving the request, the local shadow sends the message to the lambda function using the MQTT publish/subscribe protocol. After that, the lambda function analyzes the request based on UsersAttributes.json, DevicesAttributes.json, OperationAttributes.json, and EnvironmnetAttributes.json files and makes the decision.

If the request is denied, the lambda function publishes to the user's shadow update topic. When the local shadow becomes aware of this, it updates the user's phone. During this scenario, the smart oven does not receive any indication that someone is attempting to access it. In the event that the request is granted, the local shadow of the smart oven will be notified through its update topic, and the smart oven will be updated accordingly. Once the oven is turned on, it publishes a message to the shadow update topic. A notification is sent from the oven's local shadow to the lambda function, which in turn notifies the user phone's local shadow. As a final step, the shadow of the user's phone informs the user's phone that the smart oven has been turned on successfully.

3.3.4. Performance Results

In this section, we evaluate the performance of our implementation by conducting multiple test cases. We examined three different situations with three different sets of requests. Each set of requests was executed ten times to determine the average lambda processing time.

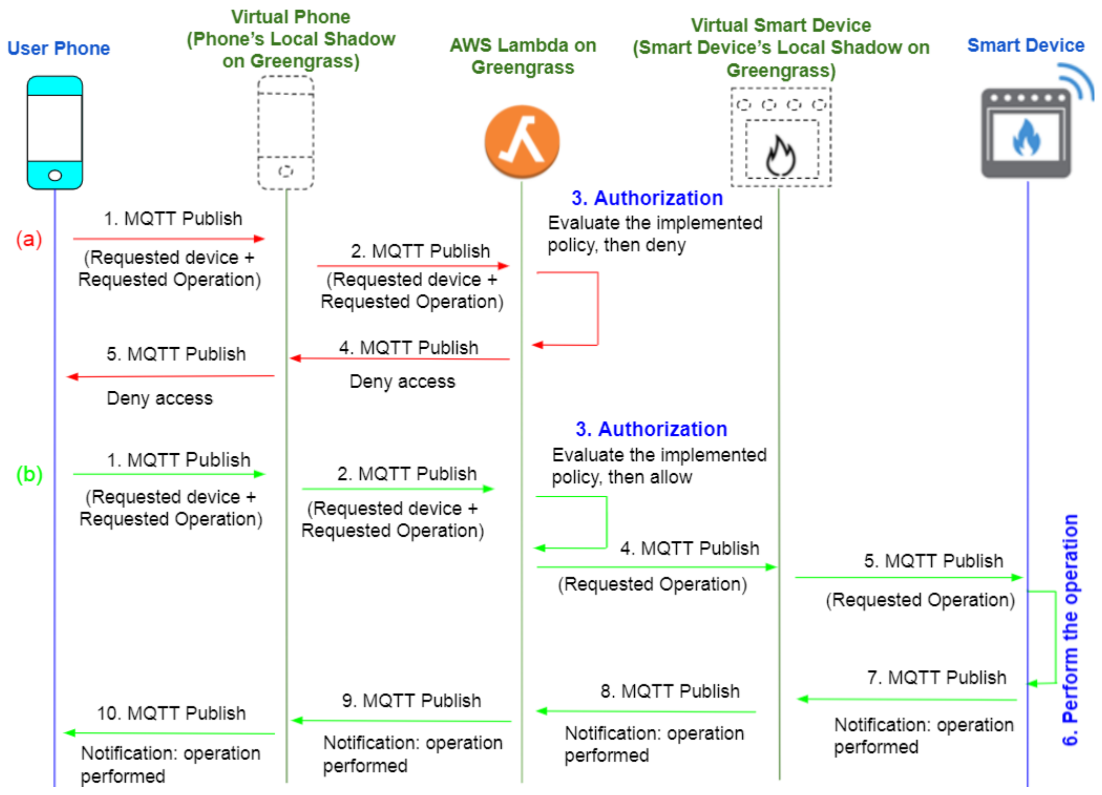
User Phone
Virtual Phone (Phone's Local Shadow on Greengrass)
AWS Lambda on Greengrass
Virtual Smart Device (Smart Device's Local Shadow on Greengrass)
Smart Device
(a)
1. MQTT Publish (Requested device + Requested Operation)
2. MQTT Publish (Requested device + Requested Operation)
3. Authorization Evaluate the implemented policy, then deny
4. MQTT Publish Deny access
5. MQTT Publish Deny access
(b)
1. MQTT Publish (Requested device + Requested Operation)
2. MQTT Publish (Requested device + Requested Operation)
3. Authorization Evaluate the implemented policy, then allow
4. MQTT Publish (Requested Operation)
5. MQTT Publish (Requested Operation)
6. Perform the operation
7. MQTT Publish Notification: operation performed
8. MQTT Publish Notification: operation performed
9. MQTT Publish Notification: operation performed
10. MQTT Publish Notification: operation performed

Figure 7. Local Request handling in our system.

Table 3 shows the average lambda function execution time measured when one user sends requests to more than one device at once. The first row shows the average lambda execution time when Bob requests to lock the front door lock. The second row shows the average lambda execution time when Bob requests to lock the front door lock, turn on the TV, and turn on the PlayStation simultaneously. Finally, the third row shows the average lambda execution time when Bob requests to lock the front door lock, open the fridge, and turn on the oven, the TV, and the PlayStation. All the requests were approved as they were supposed to according to our configured policies.

Table 3. One user sending requests to multiple devices.

Number of Users	Number of Devices	Lambda Processing Time in ms	Total Number of Requests
1	1	1.4671	10
1	3	1.33123	30 (10 per request)
1	5	1.34384	50 (10 per request)

The values presented in Table 4 represent the measured average execution time of the lambda function when multiple users are sending requests to multiple devices at the same time (one user per device). The first row describes the average execution time when the parent Bob requests to lock the front door lock. The second row shows the execution time when Bob requests to lock the front door lock, the kid Alex requests to turn on the oven, and the teenager Anne requests to open the fridge simultaneously. Additionally, the third row describes the average time when the access requests tested in the second row are repeated, the kid Suzanne requests to turn on the TV, and the teenager John requests to open the oven while one of the parents is in the kitchen. The majority of the requests were approved except for those when Alex attempted to turn on the oven, which is not allowed according to our configuration, and when Suzanne tried to turn on the television, which is not permitted according to our configuration since testing was performed on a Monday morning.

Table 4. Multiple concurrent instances of one user sending request to one device.

Number of Users	Number of Devices	Lambda Processing Time in ms	Total Number of Requests
1	1	1.4671	10
3	3	1.6925	30 (10 per request)
5	5	2.0460	50 (10 per request)

Table 5 displays the average execution time of the lambda function when multiple users simultaneously send requests to one device. The first, second, and third rows show the average time when one user (the parent Bob), three users (the parent and the two kids), and five users (the parent, the two kids, and the two teenagers) respectively all request to lock the front door lock at the same time. All the requests were denied except for when the parent Bob requested to lock the front door lock.

Table 5. Multiple users sending requests to one device.

Number of Users	Number of Devices	Lambda Processing Time in ms	Total Number of Requests
1	1	1.4671	10
3	1	1.47577	30 (10 per request)
5	1	1.55134	50 (10 per request)

4. $HABAC_\alpha$ vs. $EGRBAC$ in Terms of Theoretical Expressiveness Power

Our goal in this section is to determine if any EGRBAC configuration can be expressed fully in the $HABAC_\alpha$ model, and vice versa, and if not, which model is more expressive and in which terms. The approach used in this section is an extension of the previously published approach in [56].

4.1. From EGRBAC to $HABAC_\alpha$

Here, we introduce the $HABAC_\alpha$ configuration that translates EGRBAC configuration to be implemented by $HABAC_\alpha$ model.

In this configuration, for differentiation purposes, every EGRBAC component is followed with the suffix $_{EGRBAC}$. Similarly, every $HABAC_\alpha$ component is followed with the suffix $_{HABAC_\alpha}$. The configuration of $HABAC_\alpha$ for a given EGRBAC configuration is shown in Table 6.

Essentially, the objective is to be able to translate any EGRBAC configuration towards its equivalent $HABAC_\alpha$ configuration, so that the authorizations in the $HABAC_\alpha$ match those under EGRBAC. The inputs are EGRBAC component sets R_{EGRBAC}, U_{EGRBAC}, UA_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, EA_{EGRBAC}, DR_{EGRBAC}, $PDRA_{EGRBAC}$, P_{EGRBAC}, D_{EGRBAC}, OP_{EGRBAC}, $RPDRA_{EGRBAC}$, $DSDConstraints_{EGRBAC}$, and $SSDConstraints_{EGRBAC}$. The outputs are U_{HABAC_α}, USA_{HABAC_α}, ES_{HABAC_α}, ESA_{HABAC_α}, OP_{HABAC_α}, OPA_{HABAC_α}, D_{HABAC_α}, DA_{HABAC_α}, $UAConstraint_{HABAC_\alpha}$, $SAConstraint_{HABAC_\alpha}$, and the authorization function $Authorization_{HABAC_\alpha}(s : S_{HABAC_\alpha}, op : OP_{HABAC_\alpha}, d : D_{HABAC_\alpha}, es : ES_{HABAC_\alpha})$.

Both systems have the same users, devices, and operations. In $HABAC_\alpha$, roles are expressed using the user/session attribute $Relationship$. This attribute takes as an input a user or session and returns a set of roles that have been assigned to that user or session.

Constraints related to the static separation of duties $SSDConstraints$ are mapped into constraints related to user attributes in $HABAC_\alpha$. Similarly, constraints related to the dynamic separation of duties are mapped into session attribute constraints in $HABAC_\alpha$.

Environment roles are mapped into atomic environment-state attributes that have a range of possible values equal to $\{True, False\}$. $True$ indicates that the environment state is currently satisfied or the equivalent environment role in EGRBAC should be active. $False$ indicates that the environment state is currently not satisfied or the equivalent environment role in EGRBAC should be inactive. It is outside the scope of this model to explain how to trigger different attributes of the environment state in response to changes in the environment.

In EGRBAC, device roles are ways of categorizing permissions. In $HABAC_\alpha$, we do not have permissions. Hence, we translate device roles in EGRBAC into atomic operation attributes and atomic device attributes with a range of values equal to $\{True, False\}$. If a permission $p_x = (op_x, d_x)$, where $p_x \in P_{EGRBAC}$ is assigned to the device role dr_y, where $dr_y \in DR_{EGRBAC}$, then $da_y(d_x) = True$ and $opa_y(op_x) = True$, where $opa_y \in OPA_{HABAC_\alpha}$, $da_y \in DA_{HABAC_\alpha}$, and opa_y and da_y are the operation and device attributes which were created to be equivalent to the device role $dr_y \in DR_{EGRBAC}$.

In the final step, we construct the authorization policies. In EGRBAC, the $RPDRA$ provides specific role pairs and consequently users with access to specific device roles and therefore permissions. Therefore, an authorization function is created for each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA$. Each authorization function is then a disjunct in the final authorization function.

At the end of the translation process, there is a single user/session attribute, which is $Relationship$. The number of user attribute constraints is equal to the number of $SSDConstraints$. The number of subject attribute constraints is equal to the number of $DSDConstraints$. The number of operation attributes and the number of device attributes equals the number of device roles. The number of environment-state attributes is equal to the number of environment roles.

Table 6. EGRBAC configuration in $HABAC_\alpha$.

- $U_{HABAC_\alpha} = U_{EGRBAC}$
- $USA_{HABAC_\alpha} = \{Relationship\}$
- $Range(Relationship) = R_{EGRBAC}$
- $Relationship : u \in U_{HABAC_\alpha} \rightarrow 2^{R_{EGRBAC}}$
- $Relationship : s \in S_{HABAC_\alpha} \rightarrow 2^{R_{EGRBAC}}$
- $(\forall u_i \in U_{HABAC_\alpha})[Relationship(u_i) = \{r_x | (u_i, r_x) \in UA_{EGRBAC}\}]$
- $UAConstraint_{HABAC_\alpha} = \{uac_i\}$, where:
- $\forall(ssdc_i = (r_i, R_j) \in SSDConstraints_{EGRBAC})[uac_i = ((Relationship, r_i), UAP_j)]$, where: $UAP_j = \{(Relationship, r_n) | r_n \in R_j\}$
- $SAConstraint_{HABAC_\alpha} = \{sac_i\}$, where:
- $\forall(dsdc_i = (r_i, R_j) \in DSDConstraints_{EGRBAC})[sac_i = ((Relationship, r_i), UAP_j)]$, where: $UAP_j = \{(Relationship, r_n) | r_n \in R_j\}$
- $ES_{HABAC_\alpha} = \{Current\}$
- $ESA_{HABAC_\alpha} = ER_{EGRBAC}$
- $(\forall esa_i \in ESA_{HABAC_\alpha})[esa_i : es \in ES_{HABAC_\alpha} \rightarrow \{True, False\}]$
- $D_{HABAC_\alpha} = D_{EGRBAC}$, $OP_{HABAC_\alpha} = OP_{EGRBAC}$
- $DA_{HABAC_\alpha} = OPA_{HABAC_\alpha} = DR_{EGRBAC}$
- $(\forall da_i \in DA_{HABAC_\alpha})\ [da_i : d \in D_{HABAC_\alpha} \rightarrow \{True, False\}]$
- $(\forall opa_i \in OPA_{HABAC_\alpha})[opa_i : op \in OP_{HABAC_\alpha} \rightarrow \{True, False\}]$
- $(\forall (dr_y \in DR_{EGRBAC}, p_x \in \{p_i | (p_i, dr_y) \in PDRA_{EGRBAC}\}))[dr_y(p_x.op) = True, dr_y(p_x.d) = True]$
- Initialize the authorization function $Authorization(s : S_{HABAC_\alpha}, op : OP_{HABAC_\alpha}, d : D_{HABAC_\alpha}, current : ES_{HABAC_\alpha})$
- For each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA_{HABAC_\alpha}$, we construct an authorization function as follows:
  1. $SetOfESA = \text{“}TRUE\text{”}$.
  2. $\forall(esa \in ER_i)[SetOfESA = SetOfESA + \text{“} \wedge \text{”} + \text{“}esa(current) = True\text{”}]$.
  3. $CurrentAuth = \text{“}r_i\text{”} + \text{“} \in \text{”} + \text{“}Relationship(s)\text{”} + \text{“} \wedge \text{”} + \text{“}dr_i(op) = True\text{”} + \text{“} \wedge \text{”} + \text{“}dr_i(d) = True\text{”} + \text{“} \wedge \text{”} + \text{“}SetOfESA\text{”}$.
  4. $Authorization(s : S_{HABAC_\alpha}, op : OP_{HABAC_\alpha}, d : D_{HABAC_\alpha}, current : ES_{HABAC_\alpha}) \leftarrow Authorization(s : S_{HABAC_\alpha}, op : OP_{HABAC_\alpha}, d : D_{HABAC_\alpha}, current : ES_{HABAC_\alpha}) + \text{“} \vee \text{”} + \text{“}CurrentAuth\text{”}$.

EGRBAC grants access to a specific set of permissions through a series of assignments, as explained in Section 2.1. $RPDRA$ is the most important assignment, which assigns device roles to role pairs. To put it another way, controlling $RPDRA$ allows us to decide which role pairs and, therefore, roles are authorized to access specific device roles and, subsequently, permissions. In $HABAC_\alpha$, however, no such point of vulnerability or "attack point" can be controlled to prevent specific access. Therefore, we are unable to create something equivalent to EGRBAC $PRConstraints$ in $HABAC_\alpha$. Only by checking individual requests to access permissions and searching for prohibited users will it be possible to ensure that particular prohibited users will not be granted access to particular permissions. EGRBAC offers the advantage of the ability to enforce such constraints at the time of assignment, whereas $HABAC_\alpha$-like models would need to enforce them at the time of enforcement.

To summarize, the construction of $HABAC_\alpha$ shown in Table 6 is equivalent to the given EGRBAC configuration, including static and dynamic separation of duty constraints. Since the construction is straightforward and one-for-one, the claim of equivalence is intuitively logical. Ref. [65] can be used for a formal argument.

4.2. From $HABAC_\alpha$ to EGRBAC

Throughout this section, we will describe our methodology for generating EGRBAC components and configurations from $HABAC_\alpha$ configuration. This methodology is inspired by the approach proposed in [56], which follows a bottom-up role engineering approach. It applies for $HABAC_\alpha$ policies containing environment attributes and static user/session, operation, and device attribute functions. However, it is incapable of handling policies that compare two different types of attributes. As a result of certain limitations in EGRBAC, it is either impossible to capture policies involving dynamic user/session, operation, and device attribute functions or extremely expensive (resulting in a role explosion).

In the following sections, we describe our approach and demonstrate it step by step. Our demonstration starts from Use Case B introduced in Figure 5 and applies our methodology to construct its equivalent EGRBAC configuration.

4.2.1. From Authorization Function to Authorization Array

Here, as a preliminary step, we first transform the authorization function into a disjunctive normal form (DNF). In Figure 8, we display the authorization function of Use Case B, which is presented in Figure 5 after transforming it into a DNF format by following the standard approach.

We call each conjuncted term a condition. We have the session, environment, device, operation, and mix conditions, which receptively involve user/session, environment, device, operation, and more than one type of attribute. In Figure 8 we have six conjunctive clauses. There is one conjunctive clause per access authorization rule.

$Authorization(s : S, op : OP, d : D, es : ES) \equiv$
$(Relationship(s) = kid \wedge\ day(current) \in \{Sa, S\} \wedge \text{12:00} \leq time(current) \leq \text{19:00} \wedge KidsFriendly(op) = True) \vee$
$(Relationship(s) = kid \wedge\ day(current) \in \{M, T, W, Th, F\} \wedge \text{17:00} \leq time(current) \leq \text{19:00} \wedge KidsFriendly(op) = True) \vee$
$(Relationship(s) = teenager \wedge ParentInTheHouse(current) = True \wedge\ DangerouseDevices(d) = True) \vee$
$(Relationship(s) = teenager \wedge\ KidsFriendly(op) = False) \vee$
$(Relationship(s) = teenager \wedge\ KidsFriendly(op) = True) \vee$
$(Relationship(s) = parent)$

Figure 8. The authorization function of Use Case B in DNF format.

To build the authorization array we examine every $u_i \in U$, $d_j \in D$, and $op_k \in OP$ combination against each conjunctive clause, whenever a combination satisfies every term (condition) in a conjunctive clause except those conditions that involve environment-state attributes, where we create a row $(u_i,\ d_j,\ op_k,\ current,\ C)$ for that combination in the authorization array. C denotes the set of session and environment-related conditions in the evaluated conjunctive clause. When evaluating a combination u_i, d_j, op_k against a conjunctive clause, unsatisfying a device condition means that either the condition is not true for the evaluated device d_j, or the condition references an undefined attribute value for the evaluated device d_j. Similarly, unsatisfying an operation condition means either the condition is not true for the evaluated operation op_k, or the condition references an undefined attribute value for the evaluated operation op_k. Finally, unsatisfying a user condition means that the condition is not true for the evaluated user u_i.

Authorizations array (AA): an authorization of row $(u_i, d_j, op_k, es_l, C)$ denotes that the user u_i is allowed to perform an operation op_k on a device d_j during the environment state es_l whenever the set of environment and session conditions in C are satisfied. Table 7 provides the AA for Use Case B. Different colors represent authorization fields for different users.

Table 7. AA for Use Case B (Different colors represent authorization fields for different users).

User u	Device d	Operation op	Environment State es	Conditions C
Suzanne	iPad	$A5$	current	X
Suzanne	iPad	$A8$	current	X
Suzanne	iPad	$A5$	current	Z
Suzanne	iPad	$A8$	current	Z
Bob	iPad	$A5$	current	$\{Relationship(s) = parent\}$
Bob	iPad	$A8$	current	$\{Relationship(s) = parent\}$
Bob	iPad	$A11$	current	$\{Relationship(s) = parent\}$
Bob	iPad	$Games$	current	$\{Relationship(s) = parent\}$
Bob	iPad	$Movies$	current	$\{Relationship(s) = parent\}$
Bob	lawnMower	ON	current	$\{Relationship(s) = parent\}$
Bob	lawnMower	OFF	current	$\{Relationship(s) = parent\}$
Bob	FrontDoor	$Lock$	current	$\{Relationship(s) = parent\}$
Bob	FrontDoor	$Unlock$	current	$\{Relationship(s) = parent\}$
John	iPad	$A5$	current	$\{Relationship(s) = teenager\}$
John	iPad	$A8$	current	$\{Relationship(s) = teenager\}$
John	iPad	$A11$	current	$\{Relationship(s) = teenager\}$
John	iPad	$Games$	current	$\{Relationship(s) = teenager\}$
John	iPad	$Movies$	current	$\{Relationship(s) = teenager\}$
John	FrontDoor	$Lock$	current	Y
John	FrontDoor	$Unlock$	current	Y

X = $\{Relationship(s) = kid, day(current) \in \{Sa, S\}, 12{:}00 \leq time(current) \leq 19{:}00\}$; Y = $\{Relationship(s) = teenager, ParentInTheHouse(current) = True\}$; Z= $\{Relationship(s) = kid, day(current) \in \{M, T, W, Th, F\}, 17{:}00 \leq time(current) \leq 19{:}00\}$.

4.2.2. Approach

The goal is to construct EGRBAC elements, assignments, and derived relationships from $HABAC_\alpha$ policies so that the authorizations are the same as those under $HABAC_\alpha$. In this construction, for differentiation purposes, every EGRBAC component is followed with the suffix $_{EGRBAC}$. Similarly, every $HABAC_\alpha$ component is followed with the suffix $_{HABAC_\alpha}$.

The inputs are $HABAC_\alpha$ components U_{HABAC_α}, D_{HABAC_α}, OP_{HABAC_α}, USA_{HABAC_α}, ESA_{HABAC_α}, OPA_{HABAC_α}, DA_{HABAC_α}, and the authorization array AA. The outputs are EGRBAC components U_{EGRBAC}, R_{EGRBAC}, UA_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, EA_{EGRBAC}, RP_{EGRBAC}, $RPRA_{EGRBAC}$, $RPEA_{EGRBAC}$, D_{EGRBAC}, OP_{EGRBAC}, P_{EGRBAC}, DR_{EGRBAC}, $PDRA_{EGRBAC}$, and $RPDRA_{EGRBAC}$. The steps are following:

Step 1: Initialization. Both systems have the same set of users, devices, and operations, hence $U_{EGRBAC} = U_{HABAC_\alpha}, D_{HABAC_\alpha} = D_{HABAC_\alpha}$, and $OP_{EGRBAC} = OP_{HABAC_\alpha}$. For every operation op_i, and device d_j pair, where op_i is assigned to d_j by the device manufacturers, create a permission.

Step 2: Create the set of device roles DR_{EGRBAC}. (a) Create a device role dr for each operation attribute instance or device attribute instance. DR here are represented as a condition of the form $opa = x$, or $da = x$. Where x is an instance of the attribute value. (b) Create one device role called remaining permissions $RemPerm$ for all the permissions $p_l = (d_i, op_j)$, where d_i is not assigned to any device attributes, and op_j is not assigned to any operation attribute. This device role captures the cases where some users have access to specific permissions directly without involving the device's or operation's attributes.

Step 3: Build the permission device role assignment array $PDRA$. It is a many-to-many mapping of P_{EGRBAC} set and DR_{EGRBAC} set (constructed in Step 2). To construct $PDRA$ we first make a column for each $dr \in DR_{EGRBAC}$, and make a row for each permission $p \in P_{EGRBAC}$. Then, we fill the array $PDRA$, where $PDRA[i, j] = 1$ in two cases, first if for the permission p_i (corresponding to the row i) $p_i.op$ or $p_i.d$ satisfies the condition corresponding to the device role of the column j dr_j. Second, if $p_i.op$ is not assigned to any operation attribute, and $p_i.d$ is not assigned to any device attribute, and the device role corresponding to this column is $RemPerm$. $PDRA[i, j] = 0$ otherwise. For every $PDRA[i, j] = 1$, add the pair (p_i, dr_j) to the set $PDRA$ of EGRBAC. See Table 8 for the $PDRA$ array of Use Case B.

Step 4: Build the user device role authorization array $UDRAA$ from the authorization array AA, and $PDRA$. $UDRAA \subseteq U_{EGRBAC} \times DR_{EGRBAC}$, a many-to-many mapping between U_{EGRBAC} and DR_{EGRBAC}. To construct $UDRAA$, we first make a row for each user, and a column for each device role. Then, for every $UDRAA[i, j] \in UDRAA$ we check the AA for every u_i, and p_x combination, where $(p_x, dr_j) \in PDRA$. u_i is the user corresponding to the row i, while dr_j is the device role corresponding to the column j in $UDRAA$. Here, we have three cases: (a) $UDRAA[i, j] = 1$ if user u_i can access all the permissions assigned to the device role dr_j without any condition. (b) $UDRAA[i, j] = Y$, where Y is a set of conditions sets. Each conditions set is a set of session, and environment conditions that need to be satisfied together for a u_i to access all the permissions assigned to dr_j. Please note that these sets of conditions must be the same for each permission assigned to dr_j. (c) Finally, $UDRAA[i, j] = 0$ if user u_i is not allowed to access all the permissions assigned to the device role dr_j, or is allowed to access different permissions in dr_j but under different set of conditions. Table 9 shows $UDRAA$ for Use Case B.

Step 5: Follow the EGRBAC users and environment roles constructing algorithm presented in Section 4.2.3 to construct the rest of the EGRBAC components (R_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, RP_{EGRBAC}), assignments (UA_{EGRBAC}, EA_{EGRBAC}, $RPDRA_{EGRBAC}$), and derived relations ($RPRA_{EGRBAC}$, $RPEA_{EGRBAC}$). The set of user roles R_{EGRBAC} constructed here is the set of candidate user roles.

Step 6: Combine similar user roles. To achieve this, we run the role combining algorithm described in Section 4.2.4.

Table 8. PDRA array for Use Case B.

	DangerouseDevices = True	**DangerouseDevices = False**	**KidsFriendly = True**	**KidsFriendly = False**	**RemPerm**
$(iPad, A5)$	0	0	1	0	0
$(iPad, A8)$	0	0	1	0	0
$(iPad, A11)$	0	0	0	1	0
$(iPad, Games)$	0	0	0	1	0
$(iPad, Movies)$	0	0	0	1	0
$(lawnMower, ON)$	0	0	0	0	1
$(lawnMower, OFF)$	0	0	0	0	1
$(FrontDoor, Lock)$	1	0	0	0	0
$(FrontDoor, Unlock)$	1	0	0	0	0

Table 9. UDRAA for Use Case B.

	DangerouseDevices = True	**DangerouseDevices = False**	**KidsFriendly = True**	**KidsFriendly = False**	**RemPerm**
Suzanne	0	0	$\{X, Z\}$	0	0
Bob	$\{\{Relationship(s) = parent\}\}$	0	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$
John	$\{Y\}$	0	$\{\{Relationship(s) = teenager\}\}$	$\{\{Relationship(s) = teenager\}\}$	0

X = $\{Relationship(s) = kid,\ day(current) \in \{Sa, S\}, 12{:}00 \le time(current) \le 19{:}00\}$; Y = $\{Relationship(s) = teenager,\ ParentInTheHouse(current) = True\}$; Z= $\{Relationship(s) = kid,\ day(current) \in \{M, T, W, Th, F\}, 17{:}00 \le time(current) \le 19{:}00\}$.

4.2.3. EGRBAC Users and Environment Roles Constructing Algorithm

The goal is to construct EGRBAC elements (R_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, RP_{EGRBAC}), assignments (UA_{EGRBAC}, EA_{EGRBAC}, $RPDRA_{EGRBAC}$), and derived relations ($RPRA_{EGRBAC}$, $RPEA_{EGRBAC}$) from $UDRAA$. See Algorithm 1 for the full algorithm. The input is $UDRAA$. The outputs are R_{EGRBAC}, UA_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, EA_{EGRBAC}, RP_{EGRBAC}, and $RPDRA_{EGRBAC}$. The steps are shown as follows:

Step 1: Initialize the following EGRBAC sets $R_{EGRBAC} = \{\}$, $UA_{EGRBAC} = \{\}$, $EC_{EGRBAC} = \{\}$, $ER_{EGRBAC} = \{\}$, $EA_{EGRBAC} = \{\}$, $RP_{EGRBAC} = \{\}$, $RPDRA_{EGRBAC} = \{\}$, and the following constants m = Number of device roles, n = Number of users.

Step 2: Loop through the columns of $UDRAA$, Table 9 for Use Case B. Each column corresponds to user access rights to a specific device role. Inside each column, loop through the fields of different rows. Here we have two cases:

A. $\boldsymbol{UDRAA[i, j] = 1}$**,** according to the way $UDRAA$ was constructed, this means the user corresponding to this row u_i can access the device role of this column dr_j unconditionally. In this case, the algorithm does the following:

1. Create an environment role $er_x = Any\_Time$ and add it to the set ER_{EGRBAC}. Create an environment condition $ec_x = True$ and add it to the set EC_{EGRBAC}. Add $(\{ec_x\}, er_x)$ to the set EA, this implies that the environment role Any_Time will always be active. Create a set of environment roles SER and add er_x to it $SER = \{er_x\}$.
2. Create a role $r_m = ToString(ColumnDR(j))$ which corresponds to accessing this column device role anytime and unconditionally. Add this role to the set R_{EGRBAC}.
3. Define a role pair rp_z, where $rp_z.r = r_m$ and $rp_z.ER = SER$. Add rp_z to the set RP_{EGRBAC}.
4. Assign the role pair rp_z to the device role corresponding to this column by adding the pair $(rp_z, ColumnDR(j))$ to the set $RPDRA_{EGRBAC}$.
5. Assign the role r_m to the user corresponding to this row by adding the pair$(RawUser(i), r_m)$ to the set UA_{EGRBAC}.

B. $\boldsymbol{UDRAA[i, j] \neq 1 \wedge UDRAA[i, j] \neq 0}$**,** which means that the user u_i can access the device role dr_j under specific a set of user and environment conditions defined by $UDRAA[i, j]$. Here, $UDRAA[i, j]$ is a set of sets of conditions, where each set of conditions defines a group of session, and environment conditions that need to be satisfied together for the user u_i to be able to access the device role dr_j. Loop through each set of conditions $X \in UDRAA[i, j]$; for each X perform the following steps:

1. Loop through each condition $y \in X$. If y is an environment-state attribute condition, the algorithm creates a corresponding environment condition ec_y and adds it to the set EC_{EGRBAC}, environment role er_y and adds it to the set ER_{EGRBAC}. Adds $(\{ec_x\}, er_x)$ to the set EA_{EGRBAC}. Moreover, the algorithm adds er_y to the set of environment roles SER.
2. After looping through each condition in X, if the set SER is empty, this means this set of conditions does not contain an environment-state condition. In other words, the user of this row can access the device role of this column without any environment condition. In this case the algorithm creates an environment role $er_x = Any\_Time$ and add it to the set ER_{EGRBAC}, an environment condition $ec_x = True$ and add it to the set EC_{EGRBAC}. Add $(\{ec_x\}, er_x)$ to the set EA_{EGRBAC}, this implies that the environment role Any_Time will always be active. Add er_x to it $SER = \{er_x\}$.

3. The algorithm creates a corresponding user role $r_m = ToString(ColumnDR(j)) +'' \wedge '' + ToString(X)$ which represents accessing this column device role when the set of conditions that form X is satisfied. Add this role to the set R_{EGRBAC}.
4. Define a role pair rp_z, where $rp_z.r = r_m$ and $rp_z.ER = SER$. Add rp_z to the set RP_{EGRBAC}.
5. Assign the role pair rp_z to the device role corresponding to this column by adding the pair $(rp_z, ColumnDR(j))$ to the set $RPDRA_{EGRBAC}$.
6. Finally, assign the role r_m to the user corresponding to this row by adding the pair$(RawUser(i), r_m)$ to the set UA_{EGRBAC}.

Algorithm 1 EGRBAC Users and Environment Roles Construction

```
Require: $UDRAA$
Require: $ColumnDR(j)$: return the device role corresponding to the column j in $UDRAA$.
Require: $RawUser(i)$: return the user corresponding to the row i in $UDRAA$.
Require: $IsESC(c)$: Return True if c is an environment-state condition.
Require: $ToString(x)$ : Convert x into a string format.
1: Initilize $n$ = Number of users, $m$ = Number of device roles, $R_{EGRBAC} = \{\}$, $UA_{EGRBAC} = \{\}$,
   $EC_{EGRBAC} = \{\}$, $ER_{EGRBAC} = \{\}$, $EA_{EGRBAC} = \{\}$, $RP_{EGRBAC} = \{\}$, and $RPDRA_{EGRBAC} =$
   $\{\}$,
2: for $j \leftarrow 1$ to $m$ do
3:     for $i \leftarrow 1$ to $n$ do
4:         if $UDRAA[i,j] = 1$ then
5:             $er_x = Any\_Time, ec_x = True$
6:             $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_x\}, ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_x\}$
7:             $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_x\}, er_x)\}$
8:             $SER = \{er_x\}$
9:             $r_m = ToString(ColumnDR(j))$
10:            $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
11:            $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
12:            $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup \{(rp_z, ColumnDR(j))\}$
13:            $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
14:
15:        else if $UDRAA[i,j] \neq 1 \wedge UDRAA[i,j] \neq 0$ then
16:            for each $X \in UDRAA[i,j]$ do
17:                $SER = \{\}$
18:                for each $y \in \{y|(y \in X) \wedge IsESC(y)\}$ do
19:                    Create $ec_y$, and $er_y$
20:                    $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_y\}, ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_y\}$
21:                    $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_y\}, er_y)\}$
22:                    $SER = SER \cup \{er_y\}$
23:                end for
24:                if $SER == \{\}$ then
25:                    $er_x = Any\_Time, ec_x = True$
26:                    $EC_{RC} = EC_{RC} \cup \{ec_x\}, ER_{RC} = ER_{RC} \cup \{er_x\}$
27:                    $EA_{RC} = EA_{RC} \cup \{(\{ec_x\}, er_x)\}$
28:                    $SER = \{er_x\}$
29:                end if
30:                $r_m = ToString(ColumnDR(j)) +`` \wedge '' + ToString(X)$
31:                $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
32:                $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
33:                $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup (rp_z, ColumnDR(j))$
34:                $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
35:            end for
36:        end if
37:    end for
38: end for
```

4.2.4. Users Roles Combining Algorithm

This algorithm is designed to combine roles with similar user assignments. When two roles r_i, r_j are assigned to the same set of users, the algorithm performs the following:

1. For every role pair rp_k, in which the role part of it $rp_k.r$ is equal to r_i, change the role part of it to r_j ($rp_k.r = r_j$).
2. Remove r_i from the set of roles R_{EGRBAC}.
3. For every $(u_l, r_i) \in UA_{EGRBAC}$, remove the pair (u_l, r_i) from the set UA_{EGRBAC}.
4. For every $((r_i, ER_x), dr_y) \in RPDRA_{EGRBAC}$, remove $(r_i, ER_x), dr_y)$ from the set $RPDRA_{EGRBAC}$, and instead add the pair $((r_j, ER_x), dr_y)$ to the set $RPDRA_{EGRBAC}$. For the detailed algorithm, please refer to Algorithm 2.

Upon implementing the first five steps outlined in Section 4.2.2, we will end up with a set of nine roles, as shown in Figure 9. Different roles will be assigned to different users as follows:

$$UA = \{(suzanne, r_3), (suzanne, r_4), (bob, r_1), (bob, r_5), (bob, r_7), (bob, r_9), (john, r_2), (john, r_6), (john, r_8))\}.$$

As a result of the user role-combining algorithm, the constructed nine roles will be merged into three roles. Moreover, the user role assignment set will be composed of three pairs, as follows:

$$R = \{r_a \equiv r_1, r_5, r_7, r_9, r_b \equiv r_2, r_6, r_8, r_c \equiv r_3, r_4\}.$$

$$UA = \{(bob, r_a), (john, r_b), (suzanne, r_c), \}.$$

Algorithm 2 Users Roles Combining Algorithm

```
Require: $R_{EGRBAC}$: The set of roles
Require: $U(r)$: Returns the set of users assigned to the role r.
Require: $RP(r)$: Returns the set of role pairs associated with the role r.
 1: for each $r_i, r_j \in R_{EGRBAC}$ do
 2:     if $U(r_i) = U(r_j)$ then
 3:         for each $rp_k \in RP(r_i)$ do
 4:             $rp_k.r = r_j$
 5:         end for
 6:
 7:         $R_{EGRBAC} = R_{EGRBAC} \setminus r_i$
 8:
 9:                                         ▷ Delete all $UA$ pairs related to $r_i$
10:         for each $(u_l, r_i) \in UA_{EGRBAC}$ do
11:             $UA = UA \setminus (u_l, r_i)$
12:         end for
13:
14:                                         ▷ Replace all $RPDRA$ pairs related to $r_i$
15:         for each $((r_i, ER_x), dr_y) \in RPDRA_{EGRBAC}$ do
16:             $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \setminus ((r_i, ER_x), dr_y)$
17:             $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup$
18:                                 $\{((r_j, ER_x), dr_y)\}$
19:         end for
20:     end if
21: end for
```

$r_1 \equiv DangerouseDevices = True \wedge Relationship(s) = parent,$
$r_2 \equiv DangerouseDevices = True \wedge \{Relationship(s) = teenager,$
$\quad ParentInTheHouse(current) = True\},$
$r_3 \equiv KidsFriendly = True \wedge \{Relationship(s) = kid,$
$\quad day(current) \in \{Sa, S\}, 12{:}00 \leq time(current) \leq 19{:}00\},$
$r_4 \equiv KidsFriendly = True \wedge \{Relationship(s) = kid,$
$\quad day(current) \in \{M, T, W, Th, F\}, 17{:}00 \leq time(current) \leq 19{:}00\},$
$r_5 \equiv KidsFriendly = True \wedge Relationship(s) = parent,$
$r_6 \equiv KidsFriendly = True \wedge Relationship(s) = teenager,$
$r_7 \equiv KidsFriendly = False \wedge Relationship(s) = parent,$
$r_8 \equiv KidsFriendly = False \wedge Relationship(s) = teenager,$
$r_9 \equiv RemPerm \wedge Relationship(s) = parent$

Figure 9. Initial Set of Roles Before Running the Role Merging Algorithm.

4.2.5. The output of EGRBAC Constructing Approach on Use Case B

The output of EGRBAC role constructing algorithm for Use Case B is shown in Table 10.

The maximum number of created device roles is $O(|OPA| + |DA|)$. Since we create an environment role and an environment condition for each logical environment condition, the maximum number of environment roles and conditions is $\Omega(|ESA|)$. Finally, the maximum number of user roles is $O(2^{|SA|+|ESA|})$.

Table 10. The output of EGRBAC constructing approach on Use Case B.

(**a**) $U_{EGRBAC} = U_{HABAC_\alpha}, D_{EGRBAC} = D_{HABAC_\alpha}, OP_{EGRBAC} = OP_{HABAC_\alpha}, P_{EGRBAC} = \{(iPad, A5), (iPad, A8), (iPad, A11), (iPad, Games), (iPad, Movies), (FrontDoor, Lock), (FrontDoor, Unlock), (lawnMower, ON), (lawnMower, OFF)\}$
(**b**) $DR = \{DangerouseDevices = True,$ $DangerouseDevices = False,$ $KidsFriendly = True, KidsFriendly = False,$ $RemPerm\}.$
(**c**) $PDRA = \{((iPad, A5), KidsFriendly = True),$ $(iPad, A8), KidsFriendly = True),$ $((iPad, A11), KidsFriendly = False),$ $((iPad, Games), KidsFriendly = False),$ $((iPad, Movies), KidsFriendly = False),$ $((FrontDoor, Lock), DangerouseDevices = True),$ $((FrontDoor, Unlock), DangerouseDevices = True),$ $((lawnMower, ON), RemPerm),$ $((lawnMower, OFF), RemPerm)\}.$
(**d**) $EC = \{True,$ $ec_1 \equiv ParentInTheHouse(current) = True,$ $ec_2 \equiv day(current) \in \{Sa, S\},$ $ec_3 \equiv 12{:}00 \leq time(current) \leq 19{:}00\,,$ $ec_4 \equiv day(current) \in \{M, T, W, Th, F\}\,,$ $ec_5 \equiv 17{:}00 \leq time(current) \leq 19{:}00\}.$
(**e**) $ER = \{Any\_Time,$ $er_1 \equiv ParentInTheHouse(current) = True,$ $er_2 \equiv day(current) \in \{Sa, S\} \equiv Weekend,$ $er_3 \equiv 12{:}00 \leq time(current) \leq 19{:}00 \equiv Afternoon\ and\ Evening,$ $er_4 \equiv day(current) \in \{M, T, W, Th, F\} \equiv Weekdays,$ $er_5 \equiv 17{:}00 \leq time(current) \leq 19{:}00 \equiv Evening\}.$

Table 10. *Cont.*

(f) $EA = \{(\{True\}, Any\_Time), (\{ec_1\}, er_1)$,
$(\{ec_2\}, er_2), (\{ec_3\}, er_3), (\{ec_4\}, er_4), (\{ec_5\}, er_5)\ \}$.
(g) $R = \{r_a, r_b, r_c\}$.
(h) $UA = \{(bob, r_a), (john, r_b), (suzanne, r_c)\}$.
(i) $RP = \{(r_a, Any\_Time)$,
(r_b, Any_Time),
$(r_b, \{er_1\})$,
$(r_c, \{er_2, er_3\})$,
$(r_c, \{er_4, er_5\})\}$.
(j) $RPDRA = \{((r_a, Any_Time), DangerouseDevices = True)$,
$((r_a, \{Any\_Time\}), KidsFriendly = True)$,
$((r_a, \{Any\_Time\}), KidsFriendly = False)$,
$((r_a, \{Any\_Time\}), RemPerm)$,
$((r_b, \{er_1\}), DangerouseDevices = True)$,
$((r_b, \{Any\_Time\}), KidsFriendly = True)$,
$((r_b, \{Any\_Time\}), KidsFriendly = False)$,
$((r_c, \{er_2, er_3\}), KidsFriendly = True)$,
$((r_c, \{er_4, er_5\}), KidsFriendly = True)\}$.

5. Comprehensive Theoretical Comparison

In this section, we compare and analyze $HABAC_\alpha$ and EGRBAC against access control criteria adapted from [9]. These criteria are classified into two types: (a) Basic and main criteria. (b) Quality criteria.

5.1. Basic and Main Criteria

Six elements are included in this type of criteria. Each element will be discussed below. Table 11 summarizes this type of criteria comparison.

Table 11. Evaluating $HABAC_\alpha$ and $EGRBAC$ against basic and main criteria.

Criteria	***EGRBAC***	***HABAC***
	1. Constraints	
a. Static separation of duty	Yes	Yes
b. Dynamic separation of duty	Yes	Yes
c. P-R constraints	Yes	No
	2. Attributed-based specifications	
a. Static	Yes	Yes
b. Dynamic	No	Yes
3. Least privilege principle	Yes	Yes
4. Authentication	Positive (Close)	Positive (Close)
	5. Access administration	
a. User provisioning	Easy	Complicated
b. Policy provisioning	Complicated	Easy
6. Access review	Easy	Complicated
7. Administrative policies	Centralized	Centralized

5.1.1. Constraints

These are invariants that must be maintained. EGRBAC supports three constraints: static separation of duty, dynamic separation of duty, and permission–role constraints. $HABAC_\alpha$, however, cannot support permission–role constraints, as we will discuss in Section 6.

5.1.2. Attributed-Based Specifications

As explained in Section 3.1, we have two types of attributes, static and dynamic. Both models can support environment attributes. Furthermore, they both support static users, devices, and operations attributes. Dynamic users, devices, and operations attributes, however, are not supported by $EGRBAC$.

5.1.3. Least Privilege Principle

Under this principle, a subject of a system should only be permitted to have access to the least privileges required for performing the user duties. Both models adhere to this principle. They both include the component session, and a user belonging to several roles (in $EGRBAC$), or has different attributes corresponding to his roles in the house (in $HABAC_\alpha$) can invoke any subset of them that enables tasks to be accomplished in a session. Accordingly, a powerful user can keep some roles or attributes deactivated and activate them when necessary.

5.1.4. Authentication

Both models support positive (closed) authentication. They allow access only when there is an affirmative authorization for it and deny it in other circumstances.

5.1.5. Access Administration

Our comparison here is based on two administrative tasks, user provisioning and policy provisioning. Provisioning users is easier in RBAC-based models (including $EGRBAC$) than in ABAC-based models (including $HABAC_\alpha$). In RBAC models, user provisioning requires the administrator (the homeowner) to assign roles. Alternatively, in ABAC-based models, the administrator must configure different attribute values for newly provisioned users and devices. On the contrary, in ABAC-based model, policy provisioning only requires the addition of those policies to the authorization function. By contrast, in the RBAC-based model, this requires configuring a series of assignments as in $EGRBAC$.

5.1.6. Access Review

In RBAC-based models (such as $EGRBAC$), to calculate the maximum permission available for a user, it is sufficient to look into his roles, while this could be more complicated in ABAC-based models (such as $HABAC_\alpha$) [66].

5.1.7. Administrative Policies

To determine how administrative privileges are organized in any model, an access control administration model is required. In both models, it is assumed that the homeowner is responsible for granting or revoking permissions. Accordingly, we can conclude that they both support centralized administrative policies.

5.2. Quality Criteria

Here we have three essential criteria, as explained in the following.

5.2.1. Expressiveness and Meaningfulness

We believe that for an AC model to be expressive, it must maintain at least the following three characteristics. First, it must be formally defined to have a precise and rigorous specification of the intended behavior. Second, it must be sufficiently meaningful and expressive to support different types of constraints. Finally, The model should capture different types of static and dynamic attributes. Both models are formally defined. Moreover, they both maintain different constraints except for the permission–role constraints, which are not supported by $HABAC_\alpha$. Finally, as explained in the Attributed-based specifications criterion in Section 5.1, they both capture environment attributes and different types of static attributes. $EGRBAC$, however, does not capture the user's and device's dynamic attributes.

5.2.2. Flexibility

Several factors need to be evaluated to determine whether an AC model is flexible. Here, we identify three of them. First, the model must be flexible enough to meet the requirements of smart-home IoT. Furthermore, the model should support delegation, which is the ability for a subject to delegate some or all his privileges to another user. Additionally, the model should provide flexibility for adding new users or policies.

According to the criteria proposed in [8] for an access control model to fulfill smart-home IoT requirements, it should be dynamic, fine-grained, and suitable for constrained smart-home devices. Moreover, The model should be constructed specifically for smart-home IoT or interpreted for the smart-home domain, using appropriate use cases. The model should be demonstrated in a proof-of-concept to be credible using commercially available technology. Finally, the model should have a formal definition so that the intended behavior is precise and rigorous. As discussed in [8], $EGRBAC$ meets these criteria. However, as we discussed earlier, EGRBAC is not dynamic enough to capture dynamic attributes. On the other hand, the $HABAC_\alpha$ is dynamic, fine-grained, suitable for the constrained home environment, designed specifically for smart-home IoT, illustrated with two use-case demonstrations, has proof-of-concept implementation, and is formally defined. Hence, it meets the criteria proposed in [8].

As for delegation support, it is not feasible to determine this without an administration model for access control. In general, though, it has been demonstrated that RBAC-based and ABAC-based models are capable of delegation.

Both models are capable of provisioning new users and policies. Although new users are more easily provisioned in RBAC-based models (including $EGRBAC$) than in ABAC-based models (including $HABAC_\alpha$), it is more challenging to provision new policies in RBAC-based models than in ABAC-based models.

5.2.3. Efficiency Level and Scalability

The access control model should address two main factors regarding efficiency and scalability. The model's validity in the real world will be questionable if it cannot be expanded easily. In addition, the development of the model should not adversely affect its efficiency. To analyze these factors accurately, a more detailed study needs to be performed. Generally, however, smart-home IoT involves a relatively small number of users and devices. Furthermore, organizations of different sizes have widely adopted ABAC-based models and RBAC-based models, proving their scalability.

6. Discussion

In this paper, we present $HABAC_\alpha$. An attribute-based access control model for smart-home IoT that governs user-to-device authorization. User authentication is outside the scope of this model. The model captures the characteristics of different users, environments, operations, and devices. Furthermore, $HABAC_\alpha$ can give user access to some operations within a single device without giving them access to the entire device. Therefore, it is a fine-grained model.

We illustrated our model with two use-case scenarios and demonstrated its applicability with a proof-of-concept implementation on the Amazon Web Services (AWS) platform. Overall, our model is functional and can be easily applied. We can notice that the captured average lambda processing times are generally low. We understand that practical smart homes will have different and more complicated scenarios. A detailed performance evaluation is ultimately required by simulating a large set of smart things; however, our proof-of-concept implementation in AWS is meant to demonstrate the practical applicability and effectiveness of fine-grained security policies in the context of smart homes. Incorporating many scenarios from the real world will not result in a change in security policy evaluation. Nevertheless, we are considering the possibility of a more detailed performance analysis as an extension of this study. The expressiveness of the model is discussed in Section 5.1. However, to deploy this model for commercial use, a more general sophisticated expressiveness study should be conducted.

Furthermore, we carefully investigated $HABAC_\alpha$ against $EGRBAC$. Both models are specifically designed to meet smart-home challenges. Towards this goal, we first compared the theoretical expressiveness power of these models in Section 5.2. Then, we evaluated both models against access control models criteria adapted from [9].

In comparing the theoretical expressiveness power of $HABAC_\alpha$ and $EGRBAC$, we introduced approaches for translating $EGRBAC$ configuration into an equivalent $HABAC_\alpha$ setting so that the authorization rights are the same in both systems and vice versa. These approaches are adapted from [56]. Nevertheless, as discussed in Section 4.1, in $HABAC_\alpha$, it is not possible to create something similar to EGRBAC $PRConstraints$. Therefore, it is difficult to prevent future authorization of specific users to access particular devices or operations as this can only be done dynamically when the user is attempting to access the prohibited operation, as opposed to in EGRBAC, which permits this prevention to be implemented upon assignment. It can be challenging to determine the role structure in EGRBAC, but once it has been determined, it is straightforward to determine which users have what permissions and which users do not have future access to such privileges.

The EGRBAC construction approach is capable of handling $HABAC_\alpha$ policies containing environment attributes in addition to users/sessions, devices, and operations static attributes. However, due to certain limitations in EGRBAC, our approach cannot handle $HABAC_\alpha$ policies that deal with users, sessions, or devices with dynamic attributes. According to Section 3.1, dynamic attributes are those attributes that are rapidly changing. For instance, device temperature. According to our approach, attribute instances of the devices and permissions are translated into device roles. A device role in EGRBAC is a way to categorize permissions of different devices based on relatively static characteristics. Whenever permission is assigned to a specific device role, it remains associated with that role until an administration change is made. It is impossible to activate and deactivate device roles dynamically or activate and deactivate permissions assignments to different device roles. For instance, in $HABAC_\alpha$ we can create a device attribute $device\_temperature : d : D \longleftarrow \{Low, High\}$. We can easily configure an access policy that authorizes some users to access a device d_x only if $device\_temperature(d_x) = Low$. To do so in EGRBAC, we have two options. The first is to create two device roles, one for high temperature and another for low temperature for each device. For many devices and dynamic attributes, this option may lead to a role explosion. The second option is for those devices which have similar access conditions. We create a device role for low-temperature devices and a device role for high-temperature devices. However, there is no way to dynamically activate or deactivate devices membership in different device roles according to their temperatures. Furthermore, no mechanism in EGRBAC can dynamically activate d_x's high-temperature device role while deactivating the low-temperature device role when the temperature of dr_x is high and vice versa. A similar argument holds when we deal with dynamic user/session attributes.

In the process of constructing the EGRBAC, we did not take into account the following: (1) Policies that compare two types of attributes. (2) $HABAC_\alpha$ configurations that involve user attribute constraints and session attribute constraints. However, this may be a potential future development.

In Section 5 we conducted a comprehensive theoretical comparison between $HABAC_\alpha$ and EGRBAC. We analyzed each model against the access control model criteria adapted from [9].

Based on the above analysis, a hybrid model that incorporates $HABAC_\alpha$ and EGRBAC features is likely to be the most suitable for smart-home IoT, and probably more generally. Future directions could involve the development of a combined model that will prevent a "role explosion" while providing access authorizations that cover different users, environment, operations, or devices characteristics while maintaining the advantages of EGRBAC, such as ease of access review.

7. Conclusions

In this paper, we defined the $HABAC_\alpha$ access control model for smart homes. The model captures different user, environment, operation, and device characteristics based on a dynamic, fine-grained ABAC approach. Additionally, we presented two use-case scenarios for our model and demonstrated its applicability through a proof-of-concept implementation in Amazon Web Services. Furthermore, we provided a performance test to demonstrate how our system responds in different scenarios. Based on the evaluation, we can conclude that our model is applicable and functional.

Moreover, we assessed the theoretical expressiveness power of our model in comparison to EGRBAC [8], a dynamic contextually aware RBAC-based access control model. We accomplished this by providing approaches for the conversion of $HABAC_\alpha$ specifications into EGRBAC and vice versa. According to our findings, EGRBAC can handle relatively static attributes of devices and users as well as those of the environment but is incapable of handling relatively dynamic attributes of users and devices. However, unlike EGRBAC, it is difficult for $HABAC_\alpha$ to prevent future authorizations of specific users from accessing specific operations on particular devices.

Finally, we conducted a comprehensive theoretical comparison between EGRBAC and $HABAC_\alpha$ against previously published criteria for access control models. We concluded that a hybrid model incorporating $HABAC_\alpha$ and EGRBAC features would be most appropriate for use in IoT-enabled smart homes, as well as more broadly.

Author Contributions: Conceptualization, S.A. and R.S.; methodology, S.A. and R.S.; software, S.A. and J.B.; validation, S.A. and R.S.; formal analysis, S.A. and R.S.; investigation, S.A. and R.S.; resources, S.A., R.S. and J.B.; data curation, S.A.; writing—original draft preparation, S.A.; writing—review and editing, S.A.; visualization, S.A.; supervision, R.S.; project administration, S.A.; funding acquisition, S.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by NSF CREST Grant HRD1736209 and NSF CREST-PRF Award 2112590.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: This work is partially supported by NSF CREST Grant HRD1736209 and NSF CREST-PRF Award 2112590.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Internet of Things. Available online: https://en.wikipedia.org/wiki/Internet_of_things (accessed on 18 January 2022).
2. He, W.; Golla, M.; Padhi, R.; Ofek, J.; Dürmuth, M.; Fernandes, E.; Ur, B. Rethinking access control and authentication for the home internet of things (IoT). In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018.
3. Tilley, A. How a Few Words to Apple's Siri Unlocked a Man's Front Door. 2016. Available online: https://www.forbes.com/sites/aarontilley/2016/09/21/apple-homekit-siri-security/?sh=4f5270c862e5y (accessed on 18 January 2022).
4. Hill, K. Baby Monitor Hack Could Happen To 40,000 Other Foscam Users. 2013. Available online: https://www.forbes.com/sites/kashmirhill/2013/08/27/baby-monitor-hack-could-happen-to-40000-other-foscam-users/?sh=51b3cb2f58b5 (accessed on 18 January 2022).
5. Jia, J.; Qiu, X.; Cheng, C. Access control method for web of things based on role and sns. In Proceedings of the 2012 IEEE 12th International Conference on Computer and Information Technology, Chengdu, China, 27–29 October 2012.
6. Kaiwen, S.; Lihua, Y. Attribute-role-based hybrid access control in the internet of things. In *Asia-Pacific Web Conference*; Springer: Changsha, China, 2014.

7. Alramadhan, M.; Sha, K. An overview of access control mechanisms for internet of things. In Proceedings of the 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, Canada, 31 July–3 August 2017.
8. Ameer, S.; Benson, J.; Sandhu, R. The EGRBAC Model for Smart Home IoT. In Proceedings of the 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), Las Vegas, NV, USA, 11–13 August 2020.
9. Hasani, S.M.; Modiri, N. Criteria specifications for the comparison and evaluation of access control models. *Int. J. Comput. Netw. Inf. Secur.* **2013**, *5*, 19. [CrossRef]
10. Arias, O.; Wurm, J.; Hoang, K.; Jin, Y. Privacy and security in internet of things and wearable devices. *IEEE Trans. Multi-Scale Comput. Syst.* **2015**, *1*, 99–109. [CrossRef]
11. Ur, B.; Jung, J.; Schechter, S. The current state of access control for smart devices in homes. In Proceedings of the Workshop on Home Usable Privacy and Security (HUPS), Newcastle, UK, 24 July 2013.
12. Granjal, J.; Monteiro, E.; Silva, J.S. Security for the internet of things: A survey of existing protocols and open research issues. *IEEE Comm. Surv. Tutor.* **2015**, *17*, 1294–1312. [CrossRef]
13. Denning, T.; Kohno, T.; Levy, H.M. Computer security and the modern home. *Commun. ACM* **2013**, *56*, 94–103. [CrossRef]
14. Cui, A.; Stolfo, S.J. A quantitative analysis of the insecurity of embedded network devices: Results of a wide-area scan. In Proceedings of the 26th Annual Computer Security Applications Conference, Austin, TX, USA, 6–10 December 2010.
15. Oluwafemi, T.; Kohno, T.; Gupta, S.; Patel, S. Experimental security analyses of non-networked compact fluorescent lamps: A case study of home automation security. In Proceedings of the Learning from Authoritative Security Experiment Results (LASER) 2013, Arlington, VA, USA, 16–17 October 2013.
16. Fernandes, E.; Jung, J.; Prakash, A. Security analysis of emerging smart home applications. In Proceedings of the 2016 IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–26 May 2016.
17. Ho, G.; Leung, D.; Mishra, P.; Hosseini, A.; Song, D.; Wagner, D. Smart locks: Lessons for securing commodity internet of things devices. In Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security, Xi'an, China, 30 May–3 June 2016.
18. Fernandes, E.; Paupore, J.; Rahmati, A.; Simionato, D.; Conti, M.; Prakash, A. Flowfence: Practical data protection for emerging IoT application frameworks. In Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016.
19. Morgner, P.; Mattejat, S.; Benenson, Z. All your bulbs are belong to us: Investigating the current state of security in connected lighting systems. *arXiv* **2016**, arXiv:1608.03732.
20. Gupta, M.; Sandhu, R. Authorization framework for secure cloud assisted connected cars and vehicular internet of things. In Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies, Indianapolis, IN, USA, 13–15 June 2018.
21. Ouaddah, A.; Mousannif, H.; Elkalam, A.A.; Ouahman, A.A. Access control in the Internet of Things: Big challenges and new opportunities. *Comput. Netw.* **2017**, *112*, 237–262. [CrossRef]
22. Ferraiolo, D.F.; Sandhu, R.; Gavrila, S.; Kuhn, D.R.; Chandramouli, R. Proposed NIST standard for role-based access control. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2001**, *4*, 224–274. [CrossRef]
23. Sandhu, R.S. Role-based access control. In *Advances in Computers*; Elsevier: Amsterdam, The Netherlands, 1998; Volume 46, pp. 237–286.
24. Covington, M.J.; Moyer, M.J.; Ahamad, M. *Generalized Role-Based Access Control for Securing Future Applications*l Technical Report; Georgia Tech: Atlanta, GA, USA, 2000.
25. Zhang, G.; Tian, J. An extended role based access control model for the Internet of Things. In Proceedings of the 2010 International Conference on Information, Networking and Automation (ICINA), Kunming, China, 18–19 October 2010.
26. Barka, E.; Mathew, S.S.; Atif, Y. Securing the web of things with role-based access control. In *International Conference on Codes, Cryptology, and Information Security*; Springer: Rabat, Morocco, 2015.
27. Bandara, S.; Yashiro, T.; Koshizuka, N.; Sakamura, K. Access control framework for api-enabled devices in smart buildings. In Proceedings of the 2016 22nd Asia-Pacific Conference on Communications (APCC), Yogyakarta, Indonesia, 25–27 August 2016.
28. Liu, J.; Xiao, Y.; Chen, C. Authentication and access control in the internet of things. In Proceedings of the 2012 32nd International Conference on Distributed Computing Systems Workshops, Macau, China, 18–21 June 2012.
29. Bhatt, S.; Patwa, F.; Sandhu, R. Access control model for AWS internet of things. In Proceedings of the International Conference on Network and System Security, Helsinki, Finland, 1–23 August 2017.
30. Hu, V.C.; Kuhn, D.R.; Ferraiolo, D.F.; Voas, J. Attribute-based access control. *Computer* **2015**, *48*, 85–88. [CrossRef]
31. Jin, X.; Krishnan, R.; Sandhu, R. A unified attribute-based access control model covering DAC, MAC and RBAC. In Proceedings of the IFIP Annual Conference on Data and Applications Security and Privacy, Paris, France, 11–13 July 2012.
32. Bezawada, B.; Haefner, K.; Ray, I. Securing Home IoT Environments with Attribute-Based Access Control. In Proceedings of the Third ACM Workshop on Attribute-Based Access Control, Tempe, AZ, USA, 19–21 March 2018.
33. Mutsvangwa, A.; Nleya, B.; Nleya, B. Secured access control architecture consideration for smart grids. In Proceedings of the 2016 IEEE PES Power Africa, Livingstone, Zambia, 28 June–3 July 2016.

34. Xie, Y.; Wen, H.; Wu, J.; Jiang, Y.; Meng, J.; Guo, X.; Xu, A.; Guan, Z. Three-layers secure access control for cloud-based smart grids. In Proceedings of the 2015 IEEE 82nd Vehicular Technology Conference (VTC2015-Fall), Boston, MA, USA, 6–9 September 2015.
35. Gupta, M.; Benson, J.; Patwa, F.; Sandhu, R. Dynamic groups and attribute-based access control for next-generation smart cars. In Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, Richardson, TX, USA, 25–27 March 2019.
36. Bhatt, S.; Sandhu, R. ABAC-CC: Attribute-Based Access Control and Communication Control for Internet of Things. In Proceedings of the 25th ACM Symposium on Access Control Models and Technologies, Barcelona, Spain, 10–12 June 2020.
37. Ali, G.; Ahmad, N.; Cao, Y.; Asif, M.; Cruickshank, H.; Ali, Q.E. Blockchain based permission delegation and access control in Internet of Things (BACI). *Comput. Secur.* **2019**, *86*, 318–334. [CrossRef]
38. Ouaddah, A.; Elkalam, A.A.; Ouahman, A.A. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*; Springer: Saidia, Marocco, 2017.
39. Novo, O. Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE IoT J.* **2018**, *5*, 1184–1195. [CrossRef]
40. Dorri, A.; Kanhere, S.S.; Jurdak, R.; Gauravaram, P. Blockchain for IoT security and privacy: The case study of a smart home. In Proceedings of the 2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kona, HI, USA, 13–17 March 2017.
41. Ding, S.; Cao, J.; Li, C.; Fan, K.; Li, H. A novel attribute-based access control scheme using blockchain for IoT. *IEEE Access* **2019**, *7*, 38431–38441. [CrossRef]
42. Park, J.; Sandhu, R. Towards usage control models: Beyond traditional access control. In Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, Monterey, CA, USA, 3–4 June 2002.
43. Park, J. Usage Control: A Unified Framework for Next Generation Access Control. Ph.D. Thesis, George Mason University, Fairfax, VA, USA, 2003.
44. Zhang, X.; Parisi-Presicce, F.; Sandhu, R.; Park, J. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur. (TISSEC)* **2005**, *8*, 351–387. [CrossRef]
45. Guoping, Z.; Wentao, G. The research of access control based on UCON in the internet of things. *J. Softw.* **2011**, *6*, 724–731.
46. La Marra, A.; Martinelli, F.; Mori, P.; Saracino, A. Implementing usage control in internet of things: A smart home use case. In Proceedings of the 2017 IEEE Trustcom/BigDataSE/ICESS, Sydney, NSW, Australia, 1–4 August 2017.
47. Martinelli, F.; Michailidou, C.; Mori, P.; Saracino, A. Too long, did not enforce: A qualitative hierarchical risk-aware data usage control model for complex policies in distributed environments. In Proceedings of the 4th ACM Workshop on Cyber-Physical System Security, Incheon, Korea, 4 June 2018.
48. Malani, S.; Srinivas, J.; Das, A.K.; Srinathan, K.; Jo, M. Certificate-based anonymous device access control scheme for IoT environment. *IEEE Internet Things J.* **2019**, *6*, 9762–9773. [CrossRef]
49. Qiu, J.; Tian, Z.; Du, C.; Zuo, Q.; Su, S.; Fang, B. A survey on access control in the age of internet of things. *IEEE Internet Things J.* **2020**, *7*, 4682–4696. [CrossRef]
50. Ravidas, S.; Lekidis, A.; Paci, F.; Zannone, N. Access control in Internet-of-Things: A survey. *J. Netw. Comput. Appl.* **2019**, *144*, 79–101. [CrossRef]
51. Zhang, Y.; Wu, X. Access control in internet of things: A survey. *arXiv* **2016**, arXiv:1610.01065.
52. Yan, H.; Wang, Y.; Jia, C.; Li, J.; Xiang, Y.; Pedrycz, W. IoT-FBAC: Function-based access control scheme using identity-based encryption in IoT. *Future Gener. Comput. Syst.* **2019**, *95*, 344–353. [CrossRef]
53. Alshahrani, M.; Traore, I. Secure mutual authentication and automated access control for IoT smart home using cumulative keyed-hash chain. *J. Inf. Secur. Appl.* **2019**, *45*, 156–175. [CrossRef]
54. Sikder, A.K.; Babun, L.; Celik, Z.B.; Acar, A.; Aksu, H.; McDaniel, P.; Kirda, E.; Uluagac, A.S. Multi-user multi-device-aware access control system for smart home. *arXiv* **2019**, arXiv:1911.10186.
55. Sandhu, R.; Ferraiolo, D.; Kuhn, R. The NIST model for role-based access control: Towards a unified standard. In Proceedings of the ACM Workshop on Role-Based Access Control, Berlin, Germany, 26–28 July 2000.
56. Ameer, S.; Sandhu, R. The HABAC Model for Smart Home IoT and Comparison to EGRBAC. In Proceedings of the ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SAT-CPS), Online, 28 April 2021.
57. Partial Function. Available online: https://en.wikipedia.org/wiki/Partial_function (accessed on 20 January 2022).
58. Atomic Sentence. Available online: https://en.wikipedia.org/wiki/Atomic_sentence (accessed on 20 January 2022).
59. Geneiatakis, D.; Kounelis, I.; Neisse, R.; Nai-Fovino, I.; Steri, G.; Baldini, G. Security and privacy issues for an IoT based smart home. In Proceedings of the 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 22–26 May 2017.
60. AWS-IoT. Available online: https://aws.amazon.com/iot/ (accessed on 20 January 2022).
61. AWS IoT Greengrass. Available online: https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html (accessed on 20 January 2022).
62. AWS IoT Device SDK for Python. Available online: https://docs.aws.amazon.com/greengrass/latest/developerguide/IoT-SDK.html (accessed on 20 January 2022).
63. MQTT.fx—A JavaFX based MQTT Client. Available online: https://softblade.de/en/welcome/ (accessed on 20 January 2022).

64. The Transport Layer Security (TLS) Protocol. Available online: https://tools.ietf.org/html/rfc5246 (accessed on 20 January 2022).
65. Tripunitara, M.; Li, N. A theory for comparing the expressive power of access control models. *J. Comput. Secur.* **2007**, *15*, 231–272. [CrossRef]
66. Kuhn, D.R.; Coyne, E.J.; Weil, T.R. Adding attributes to role-based access control. *Computer* **2010**, *43*, 79–81. [CrossRef]