USER-TO-DEVICE ACCESS CONTROL MODELS FOR CLOUD-ENABLED IOT WITH SMART HOME CASE STUDY

by

SAFWA AMEER, M.Sc.

DISSERTATION
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

COMMITTEE MEMBERS:
Prof. Ravi Sandhu, Ph.D., Chair
Prof. Jianwei Niu , Ph.D.
Prof. Xiaoyin Wang, Ph.D.
Prof. Weining Zhang, Ph.D.
Prof. Ram Krishnan, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Science
Department of Computer Science
August 2021

DEDICATION

I would like to dedicate this dissertation to my mother Dr. Amira Burhan, and my father Dr. Ameer Adil, who helped me in all things great and small. This work is also dedicated to my husband Mr. Tareg El-sherif, who motivated and encouraged me through out the most challenging phase of my life. Finally, I dedicate this work to my lovely prince Ziyad and my little princess Yasmine who are my source of strength.

ACKNOWLEDGEMENTS

First, I would like to express my sincere gratitude and respect to my advisor and mentor Professor Ravi Sandhu for the continuous support through this demanding journey, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. I could not have imagined having a better advisor and mentor for my PhD study. The knowledge and experience I gained professionally and personally from him will be remembered, and will set course for my career and life.

A big thank you to Mr. James Benson. I am very fortunate to have worked with him. Mr. James Benson is an outstanding technology research associate at ICS. His technical support was absolutely needed to sail through this journey. I am privileged to work with the best research team and the staff at ICS which together brought me in a position to write my Ph.D. dissertation.

My sincere thanks also go to my committee members, Professor Jianwei Niu, Dr. Weining Zhang , Dr. Ram Krishnan , and Dr. Xiaoyin Wang for serving as my committee members, for their thoughtful comments, suggestions and time. Their observations have made this research work more valuable and interesting.

I would like to acknowledge the faculty members of the Computer Science department for their wisdom and support. I would like to thank the staff members Suzanne Tanaka, Susan Allen, and others from the ICS and the CS department for their tremendous kindness, help, and support.

A special thanks to my family. Words cannot express how grateful I am to my role model my mother Dr. Amira Burhan, father Dr. Ameer Adil, my beloved and supportive husband Tareg Elsherif, my brothers, and my sister for supporting me throughout writing this dissertation and my life in general. Your prayers for me was what sustained me thus far. A special thanks to my uncle Professor Hatim Sharif for his assistance and guidance throughout my career. At the end, I would like to thank all my friends who supported and motivated me to strive towards my goal.

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements

explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this Masters Thesis/Recital Document or Doctoral Dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin, and legibility requirements. In such cases, connecting texts, which provide logical bridges between different manuscripts, are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the students contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the Masters Thesis/Recital Document or Doctoral Dissertation attest to the accuracy of this statement.

August 2021

USER-TO-DEVICE ACCESS CONTROL MODELS FOR CLOUD-ENABLED IOT WITH SMART HOME CASE STUDY

Safwa Ameer, Ph.D.
The University of Texas at San Antonio, 2021

Supervising Professor: Prof. Ravi Sandhu, Ph.D.

The Internet of Things (IoT), sometimes called the Internet of Everything, is a new technology paradigm envisioned as a global network of physical objects (things) that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet . The concept of IoT has attracted many application domains including consumer applications (smart homes, elder care), organizational applications (medical and health care, vehicular communication systems), industrial applications (manufacturing, agriculture), infrastructure applications (smart cities, energy management), and military applications (Internet of Battlefield Things). Soon IoT will affect all industries and everyone's daily life. This requires usable authentication and sophisticated access control specification mechanisms that are currently lacking. It is widely recognized that the potential benefits of IoT can be fully realized only in combination with cloud computing. Cloud support enables long-term storage of the massive amounts of data produced by IoT devices and the compute-intensive analysis of this data to improve overall operations, especially in context of limited capacity of IoT devices. Moreover, cloud support enables data and analytical sharing. Hence the concept of cloud-enabled IoT (also sometimes called cloud-assisted IoT).

In rapidly evolving IoT domains, security and privacy of data and information is always at considerable risk from unauthorized actors and malicious attackers. One of the critical security services in IoT that mostly all researchers agree upon is access control (AC). Insecure access to web, backend APIs (Application Programming Interfaces), cloud and mobile interfaces are among the top vulnerabilities for IoT applications. However, commercial IoT frameworks fall short in implementing access control to these interfaces.

Soon IoT will be part of every home turning our houses into smart houses, in which we have multiple users with complex social relationships between them using the same smart devices. Providing an appropriate access control model for home IoT services is a vital but challenging topic. Authorization issues have been explored extensively for many different domains. However, home IoT is significantly different from traditional domains which necessitate a rethinking of access control and authentication. This dissertation investigates user-to-device access control requirements in home IoT, and then develops and demonstrates four different access control models for user-to-device interaction in smart home IoT.

First, it proposes the extended generalized role-based access control ($EGRBAC$) model for smart home IoT. It provides a formal definition for $EGRBAC$ and illustrates its features with a use case. A proof-of-concept demonstration utilizing AWS-IoT Greengrass is also discussed.

Second, it introduces the smart home IoT attribute-based access control model ($HABAC$). It provides a formal definition for $HABAC$ and illustrates its features with a use case. It provides an analysis of $HABAC$ relative to $EGRBAC$. It compares the theoretical expressive power of these models by providing algorithms for converting an $HABAC$ specification to $EGRBAC$ and vice versa. Moreover, it discusses the insights for practical deployment of these models resulting from these constructions. This dissertation identifies the need for a combined (role-based and attribute-based) access control model for smart home IoT.

Third, this dissertation develops a formal role-centric hybrid access control model ($HyBAC_{RC}$). It further demonstrates the features of $HyBAC_{RC}$ through a use case scenario, and a proof of concept implementation.

Fourth, it introduces an attribute-centric hybrid access control model ($HyBAC_{AC}$). It formally defines the model, and illustrates its features with a use case scenario and a proof of concept implementation. It analyzes this model relative to $HyBAC_{RC}$, $HABAC$ and $EGRBAC$ models. Moreover, it provides approaches for converting an $HyBAC_{RC}$ specification to $HyBAC_{AC}$ and vice versa. It argues that a role-centric hybrid access control model combining ABAC and RBAC features may be the most suitable for user-to-device smart home IoT access control.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The Internet of Things (IoT), sometimes called the Internet of Everything or the Industrial Internet, is a new technology paradigm envisioned as a global network of physical objects (things) that are embedded with sensors, software, and other technologies for the purpose of connecting and exchanging data with other devices and systems over the Internet [9, 73]. Currently, IoT is one of the most talked about topics in technology. It has already become an indispensable component of our lives. It has attracted many applications including consumer applications (smart homes, elder care, etc.), organizational applications (medical and health care, vehicular communication systems, etc.), industrial applications (manufacturing, agriculture, etc.), infrastructure applications (smart cities, energy management, etc.), and military applications (Internet of Battlefield Things). It is widely recognized that the potential benefits of IoT can be fully realized only in combination with cloud computing. Cloud support enables long-term storage of the massive amounts of data produced by IoT devices and the compute-intensive analysis of this data to improve overall operations, especially in the context of the limited capacity of IoT devices. Moreover, cloud support enables data and analytical sharing. Hence the concept of cloud-enabled IoT (also sometimes called cloud-assisted IoT).

One of the most popular domains for deploying smart connected devices is the smart home. A "smart home" can be defined as a residence equipped with a network of physical objects (things) that are embedded with sensors, softwares, and other technologies to connect and exchange data with other devices, users, and systems over the Internet. Smart homes main purposes are to anticipate and respond to the needs of the occupants, working to promote their comfort, convenience, security and entertainment through the management of technology within the home and connections to the world beyond [15].

Access control is concerned with limiting the activity of legitimate users [102]. It defines who has access to what, when and in which conditions. In other words, access control is any mechanism by which a system grants or revokes the right to access some data or perform some action. In

general, access control requires both authentication and authorization techniques. Authentication is any process by which a system verifies the identity of a user who wishes to access the system. Authorization determines what an authenticated user can or cannot do in the system. Our focus in this dissertation is on authorization (also called access control) while authentication is outside the scope.

Soon IoT will be part of every home turning our houses into smart houses, in which we have multiple users with complex social relationships between them using the smart devices. Nevertheless, surprisingly little attention has been paid to access control policy specification and enforcement in home IoT [59]. As illustrated in [59], the characteristics that make IoT distinct from prior computing domains necessitate a rethinking of access control and authentication.

In this research, we studied the literature on IoT access control models, analyzed it, and formulated criteria that need to be satisfied in smart home access control models. Moreover, for IoT access control models that govern user to device access, we analyzed them against our criteria. Notably, no prior model satisfies all desired specifications.

Our goal is to address the lack of widely adopted access control models for smart home IoT. We believe the best approach for this purpose is to develop a family (or series) of models ranging from relatively simple and complete to incorporating increasingly sophisticated and comprehensive features. Developing such a family has been successful in the past, most notably in the seminal role-based access control (RBAC) models of [48, 101]. Other access control model families have been published in a variety of contexts including usage control [92], role-based delegation [24], on-line social networks [34,49], attribute-based access control (ABAC) [66] and relationship-based access control [13]. The complexities of the smart home IoT environment similarly merits development of a suitable family of access control models.

We followed four different approaches to develop four different access control models for smart home IoT. Each model is formally defined and illustrated with use cases and proof of concept implementation. Finally, we analyze and compare these models. Here, one might argue that why just the home environment, how about the general IoT context? As we mentioned earlier, smart

homes have unique characteristics, hence, they require a special access control model. However, this model can be altered, and adjusted to conform with the access control specifications of other IoT domains.

1.1 Motivation

Soon IoT will affect all industries and everyone's daily life. This requires usable authentication and sophisticated access control specification mechanisms that are currently lacking.

In rapidly evolving IoT domains, stored data and information are always at considerable risk from unauthorized actors and malicious attackers. Security and privacy in IoT are primary factors that will enable wide adoption of IoT, especially at the consumer level. It is generally accepted by most researchers that access control is a critical service in IoT.

Providing an appropriate access control model for IoT services is a vital but challenging topic. Indeed, authentication and authorization issues have been intensively investigated through existing protocols for use cases outside constrained environments. The deployment of resource constrained devices along with the adoption of a plethora of technologies enlarges the attack surface and introduces new security vulnerabilities [89, 97]. Insecure access to the web, backend APIs (Application Programming Interfaces), cloud, and mobile interfaces are among the top vulnerabilities for IoT applications. Smart devices are typically configured and controlled via vendor apps, which can have a smartphone-based interface and a web-based interface through a service running on a cloud infrastructure. Services expose a web-API that allows to query and control user data and devices from the same vendor and other compliant devices from other vendors. Services from vendors can be composed with third party services e.g. Facebook, Instagram using IFTTT web service. In this complex IoT ecosystem access control should be enforced at each of these interfaces. However, commercial IoT frameworks fall short in implementing access control to these interfaces [97].

The shortcoming in applying access control policies in IoT applications leads to devices and apps being easily exploited to gain unauthorized access to devices and to users and devices data [44, 59, 89]. The need arises for dynamic and fine-grained access control mechanisms, where

users/resources are constrained [89].

To date, IoT security and privacy research has focused on such devices' insecure software-engineering practices, improper information flows, and the inherent difficulties of patching networked devices. Surprisingly little attention has been paid to access control policy specification, or authentication in home IoT [59]. Real world examples of the shortcomings of current access control policy specification and authentication for home IoT devices have begun to appear as described in [59], [114], and [61]. On the other hand, authorization issues have been explored extensively for many different domains. However, home IoT is significantly different from traditional domains such as enterprise, electronic commerce, and web services in three main ways:

1. In home IoT we have many users who use the same device, for example a smart door lock. Widely deployed techniques for specifying access-control policies and authenticating users fall short when multiple users share a device [59].

2. House residents usually have complex social relationship between them, which introduce a new threat model, e.g. a annoying child trying to control the smart light in a sibling's room, a current or ex-partner trying to abuse one or all house residents [59, 78].

3. The majority of IoT devices do not have screens and keyboards making them hands free for convenience while making authentication and access control more challenging.

The characteristics that make IoT distinct from prior computing domains necessitate a rethinking of access control and authentication [59]. In particular, the need arises for a dynamic and fine-grained access control mechanism, where users and resources are constrained [89].

1.2 Problem Statement and Solution Approach

In the literature, several access control models have been proposed for IoT in general. The majority of them are built on ABAC or RBAC. Some researchers argue that RBAC is more suitable for IoT since it is simpler in management and review, while ABAC is complex [17, 67, 68]. On the other

hand, others argue that ABAC models are more scalable and dynamic, since they can capture different devices and environment contextual information [26, 27, 121].

Hence, when it comes to smart homes, at this point it is not fully clear what is the benefit of ABAC over RBAC, or vice versa. Our intuitive insight is that a hybrid model will better capture smart home IoT access control requirements as this was already the case for traditional access control application domains. In order to further investigate this intuition our approach is to develop pure RBAC and pure ABAC based models explicitly defined to meet smart home challenges, and compare their benefits and drawbacks. This comparison will provide insights to guide us in designing adequate hybrid models.

1.3 Thesis Statement

The established paradigms of role-based and attribute-based access control can be utilized, adapted, and extended to provide fine-grained and dynamic authorization approaches for user to device access in smart home IoT. A detailed analysis of these approaches, their formal models, and implementation can eventually be utilized to develop hybrid access control models that combine role-based and attribute-base access control features to meet smart home IoT challenges.

1.4 Scope and Assumption

In smart houses we recognize two types of adversaries [59]. First an outsider hacker who is trying to get digital or physical access to the house by exploiting system vulnerabilities. Second the household members themselves, that is insiders who have legitimate digital and physical access to the house, such as family members, guests, and workers. The intention for legitimate user to break down the access control system of the smart home may vary from curiosity (e.g. a kid playing with oven setting), disturbing other family members (e.g. a kid locking his brothers outside the house), to disobedience (e.g. a kid is trying to watch TV outside the allowed entertainment time), or robbery (e.g. a worker getting access to the camera system and adjust it to shutdown at a certain time). Making sure that those legitimate users get access only to what they are authorized to by

the house owner, is the central focus of our paper. We emphasize that authorized insiders who try to hack the access control system, or to break the IoT devices to get an unauthorized access to the system are outside the scope of our threat model.

Some of the assumptions taken during this work are as follows:

- All developed four models $EGRBAC$, $HABAC$, $EGRBAC_{RC}$, and $EGRBAC_{AC}$ assume that enforcing constraints should be imposed by the administrative model and is outside the scope of the developed operational models.
- The $EGRBAC$ model assumes that environment conditions are determined by sensors deployed in the smart home under home owner control. How each sensor get triggered an hence activate its corresponding environment condition is outside the scope of this model.
- The $HABAC$ and $HyBAC_{AC}$ models assume that dynamic attribute functions are determined by sensors deployed in the smart home under home owner control. The mechanism by which these attributes get triggered is outside the scope of the access control model.
- $HyBAC_{RC}$ model assumes that environment conditions, dynamic attribute functions are determined by sensors deployed in the smart home under the control of the home owner. Triggering those sensor mechanisms are outside the scope of the model.

1.5 Summary of Contributions

Figure 1.1 presents an overview of the contributions of this dissertation as discussed below.

1. This dissertation analyzes IoT access control models proposed in the literature, and smart home IoT access control challenges to formulate criteria that need to be maintained in future proposed access control models. Furthermore, it evaluates literature on IoT access control models against this criteria.
2. This dissertation presents $EGRBAC$, which is a formally defined RBAC based model for smart home IoT access control. Furthermore, it illustrates the model with a use case scenario,

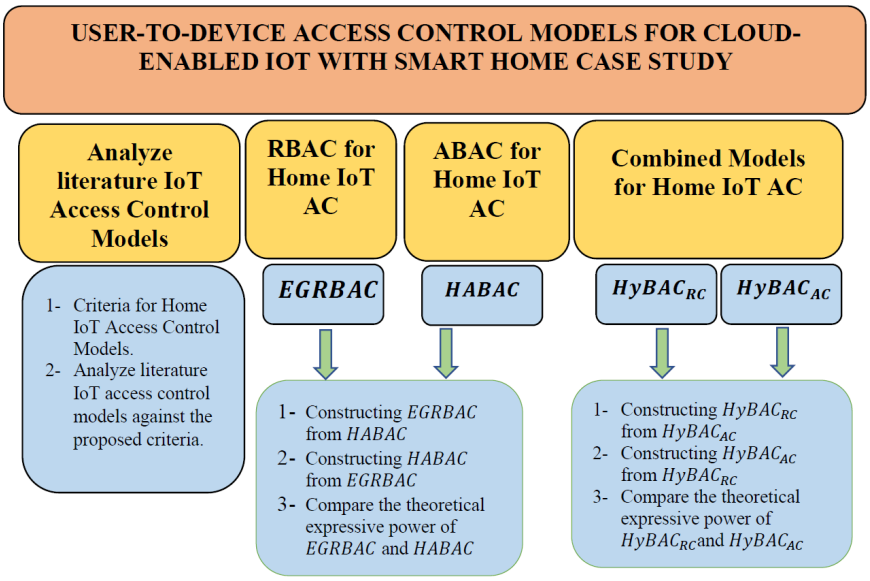

Figure 1.1: Overview of Contribution

and demonstrates its applicability with a proof of concept implementation.

3. This dissertation proposes $HABAC$, an ABAC based access control model for smart home IoT. It illustrates the model with a use case scenario.

4. This dissertation proposes approaches for constructing $EGRBAC$ specification from $HABAC$ specification and vice versa. It analyzes $EGRBAC$ relative to $HABAC$. It compares the theoretical expressive power of these models, and discusses the insights for practical deployment of these models.

5. This dissertation proposes two hybrid models for smart home IoT access control ($HyBAC_{RC}$ and $HyBAC_{AC}$). While $HyBAC_{RC}$ is a role-centric combined ABAC and RBAC model, $HyBAC_{AC}$ is an attribute-centric combined ABAC and RBAC model. These models meet smart home IoT access control requirements. Moreover, for each model, this dissertation provides a formal definition, use case demonstration, and a proof of concept implementation.

6. This dissertation proposes approaches for constructing $HyBAC_{RC}$ specification from $HyBAC_{AC}$ specification and vice versa. It analyzes $HyBAC_{RC}$ relative to $HyBAC_{AC}$. It compares the

theoretical expressive power of these models, and discusses the insights for practical deployment of these models.

1.6 Organization of the dissertation

The rest of the dissertation is organized as follows. Chapter 2 provides a brief background on the topics which form a foundation for this dissertation. Moreover, it discusses relevant related work in the literature. Chapter 3 proposes a criteria for home IoT access control models. It also introduces the extended generalized role based access control (EGRBAC) model for smart home IoT. It further demonstrates the model through a use case scenario and a proof of concept implementation. Chapter 4 introduces the smart home IoT attribute-based access control model (HABAC). Furthermore, it provides an analysis of HABAC relative to the previously EGRBAC. It compares the theoretical expressive power of these models by providing algorithms for converting an HABAC specification to EGRBAC and vice versa. Chapter 5 presents two hybrid models for smart home IoT access control. It formally defines these models, and illustrates their features through a use case scenario demonstration. A proof of concept implementation for each model is also provided. Chapter 5 compares the theoretical expressive power of these two hybrid models by providing approaches for converting between the models. Finally, it provides a comprehensive theoretical comparison between the four smart home IoT access control models introduced in this dissertation. Finally, Chapter 6 summarizes and concludes the paper. It also provides a brief overview of potential work.

CHAPTER 2: BACKGROUND AND LITERATURE REVIEW

This chapter discusses related work along with concepts and required background of prior research relevant to this dissertation. First it highlights the widely adopted role-based access control model (RBAC), and attribute-based access control model (ABAC). Moreover, it investigates the concept of combining ABAC and RBAC models, and the suggested combining approaches in the literature. Finally, related work in IoT access control is briefly discussed.

2.1 Access Control Models

Access control (sometimes referred as authorization) is concerned with limiting the activity of legitimate users [102]. It defines who has access to what, when and in which conditions. In other words, access control is any mechanism by which a system grants or revokes the right to access some data or perform some actions. The Common Criteria defines an organizational security policy as: a set of security rules, procedures, or guidelines imposed (or presumed to be imposed) now and/or in the future by an actual or hypothetical organization in the operational environment [60]. Such an organizational security policy usually relies on an access control policy [60]. An access control model is often used to rigorously specify and reason on the access control policy (e.g., to verify its consistency). However, the model does not specify how the security policy is enforced. The enforcement should be realized by technical security mechanisms, such as credentials, cryptographic transformations (e.g., signature, encryption), access control lists (ACLs), and firewalls among others [89].

Three most significant and widely known access control models are Discretionary Access Control (DAC) [102], Mandatory Access Control (MAC) [102], and Role-Based Access Control (RBAC) [**?**, 100, 101]. While each of them has their advantages, they also have shortcomings.

In DAC, owners of objects control access of users to the objects. When a user requests access to an object, the request is checked against the defined policy for that object from the owner which determines either to grant or deny the access request. Access control list (ACLs), capabilities

and relations are some ways to implement DAC. It is simple and straightforward model but has inherent weaknesses, such as copying problem and Trojan horses which can be easily exploited to gain unauthorized access to sensitive information.

MAC or lattice-based access control (LBAC) model overcomes this underlying limitation of DAC. It was specifically designed for military applications where confidentiality is the main concern. It has strict information flow policies that assigns subjects and objects with security levels which restricts unauthorized information flow. The security levels or sensitivity for objects reflects the kind of information they have and its impact after unauthorized disclosure, whereas the security clearance for subjects is based on their ranking and trustworthiness in the enterprise. A user is allowed to operate or access the information in the objects if certain predefined relationships (or properties) are satisfied by the two security levels in question.

Both DAC and MAC are based on fixed, predetermined policies. Whereas, RBAC is a more flexible and administrative friendly access control model [30]. It determines accesses based on roles assigned to the users and permissions associated with these roles on specific objects. It is the most popular access control model in the industry. However, it also has some well-known limitations such as role explosion and role-permission explosion [96]. Motivated by the limitations of the traditional access control models, attribute-based access control (ABAC) has recently received significant attention in the literature [30]. Some other access control models proposed include provenance based access control (PBAC) [91], relationship based access control (ReBAC) [34], and Capability-based access control (CapBAC) [64]. PBAC uses the provenance information attached with the underlying data, which offers utilities as usage information, versioning or pedigree information, to provide access controls. ReBAC [34] enables permissions based on the relationship among users and primarily used in online social networks (OSNs). Besides user to user (U2U) relationship, user to resource (U2R) and resource to resource (R2R) relation have also been used to control usage and administrative activities of the users in OSNs. CapBAC utilizes the concept of capability as a token, ticket, or key that gives the possessor permission to access an entity or object in a computer system.

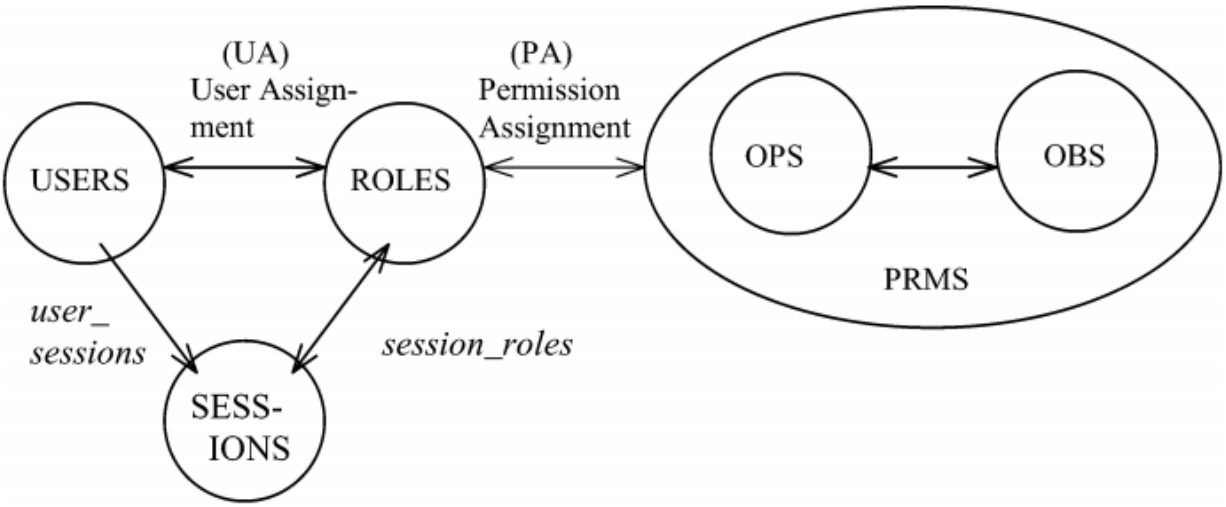

Figure 2.1: The Core RBAC Model [48]

In the following two subsections we give brief overview of RBAC and ABAC since they are central components in this dissertation.

2.1.1 Role-Based Access Control (RBAC)

The basic concept of role based access control (RBAC) model [48, 100] is that permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions. operation. In this approach, an administrator creates roles that represent specific tasks and assigns permissions to those roles (permission-role assignment), and these roles are then assigned to users (user-role assignment). RBAC is capable of enforcing both DAC and MAC models [87]. Major cloud platforms such as Amazon AWS [3], Microsoft Azure [10], and OpenStack [1] utilize role-based access control model as their authorization foundation.

The NIST standard of RBAC [48] comprises of these four components: Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. Amongst them, Core RBAC model with five basic elements (users (USERS), roles (ROLES), objects (OBS), operations (OPS), and permissions (PRMS)) is shown in Figure 2.1. RBAC is policy-neutral [101], auditable, offers permission and user-level abstraction, and provides operational and administrative scalability through roles. However, it also has some well-known limitations such as

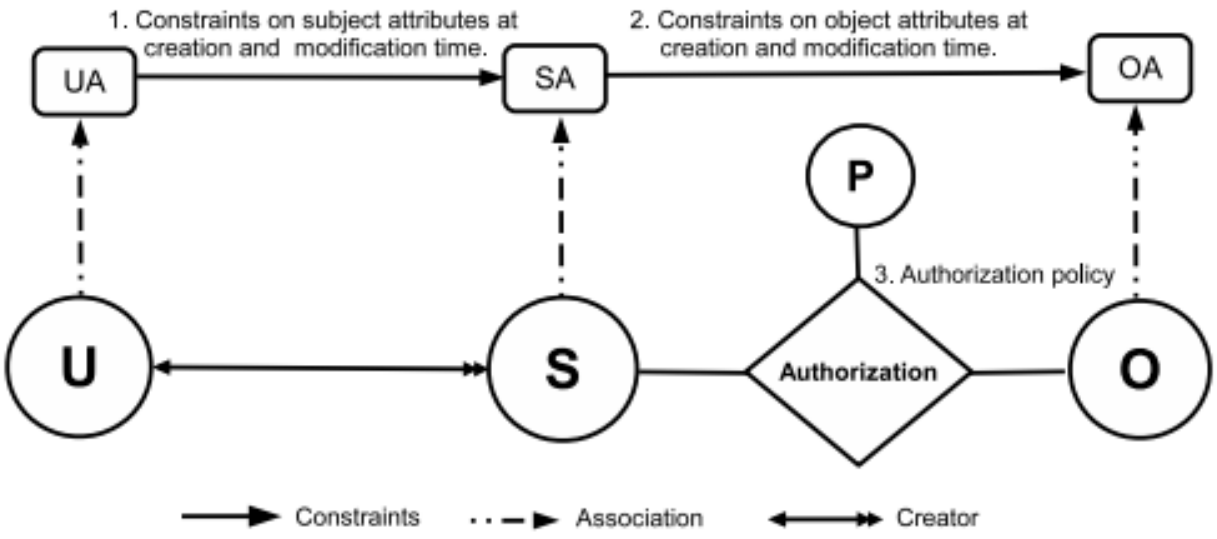

Figure 2.2: $ABAC_\alpha$ Model Components [66]

role explosion and role-permission explosion [96]. Furthermore, in RBAC, access control policies can only be defined on basis of roles which restricts access control flexibility.

2.1.2 Attribute-Based Access Control (ABAC)

RBAC has been a de facto standard access control model for over the last quarter century. However, motivated by the limitations of the traditional access control models, attribute-based access control (ABAC) has recently received significant attention in the literature [30]. The basic idea of attribute-based access control is to employ attributes (characteristics or properties) of different entities to make access control decisions regarding a subject's (e.g., user, process, etc.) access on an object (e.g., file, printer, database, etc.) in a system. The access control decisions are evaluated based on authorization policies specified by an administrator using a policy specification language. Authorization policies are defined using policy language which includes attributes that the users and objects should have to satisfy the policy and determine access control decision. Moreover, to further fine grain the access decision, other contextual and environment attributes can be used.

The concept of ABAC has been around for close to three decades in the literature with attribute based access control models designed for specific applications [106, 117] and attribute-based encryption [52, 83] for securely sharing objects or data. However, recently academia and standards

bodies like National Institute of Standards and Technology (NIST) have gained interest in ABAC and are considering it as the Next generation Access Control (NGAC) [47]. Jin et al [66] proposed a unified ABAC model called $ABAC_\alpha$ which can be configured to enforce DAC, MAC and RBAC models. Figure 2.2 shows the $ABAC_\alpha$ model. The core components are users (U), subjects (S), objects (O), user attributes (UA), subject attributes (SA), object attributes (OA), permissions (P), authorization policies, and constraint checking policies for creating and modifying subject and object attributes.

2.1.3 Combining ABAC and RBAC

Table 2.1: Combination Strategies and Options for Integrating Attributes with RBAC [70]

Option	U	R	A	Model	Permission Mapping
0	0	0	0	undefined	-
1	0	0	1	ABAC-basic	$A_1, ..., A_n \rightarrow perm$
2	0	1	0	undefined	-
3	0	1	1	ABAC-RBAC hybrid	$R, A_1, ..., A_n \rightarrow perm$
4	1	0	0	ACLs	$U \rightarrow perm$
5	1	0	1	ABAC-ID	$U, A_1, ..., A_n \rightarrow perm$
6	1	1	0	RBAC-basic	$U \rightarrow R \rightarrow perm$
7	1	1	1	RBAC-A, dynamic roles	$U, A_1, ..., A_n \rightarrow R \rightarrow perm$
8	1	1	1	RBAC-A, attribute-centric	$U, R, A_1, ..., A_n \rightarrow perm$
9	1	1	1	RBAC-A, role-centric	$U, R, A_1, ..., A_n \rightarrow perm$

The authors in [70] proposed different approaches for combining ABAC and RBAC by adding attributes to role-based access control policies. The main purpose was to combine the two models advantages, and to overcome their limitations. Table 2.1 summarizes possible combination strategies and options for integrating attributes with RBAC suggested by [70]. Options 0 and 2 are undefined but included for completeness; options 1 and 3, which have no user ID in the access decision, might appear unusual but could be used in public facilities where attributes or roles determine anonymous users' access. Options 0 and 2 are undefined but included for completeness;

options 1 and 3, which have no user ID in the access decision, might appear unusual but could be used in public facilities where attributes or roles determine anonymous users' access.

Broadly speaking, there are three RBAC-A approaches to handle the relationship between roles and attributes, all retaining some of the administrative and user permission review advantages of RBAC while allowing the access control system to work in a rapidly changing environment: dynamic roles, attribute-centric, and role-centric. Dynamic roles uses user and context attributes for assigning roles to users dynamically, similar to attribute-based user-role assignment. Attribute-centric approach assume roles as another attribute of users. Role-centric approach curtails the permissions of a role based on user attributes

2.2 IoT Access Control Models

IoT has been extensively studied by security experts. Many researchers have focused on identifying IoT security and privacy vulnerabilities [21, 37, 39, 53, 86, 107, 116]. Moreover, to analyze IoT security challenges and security design issues in specific, many researchers have conducted studies of IoT frameworks (e.g. [44, 45, 57, 62, 80, 86]). One of the critical security services in IoT that mostly all researchers agree upon is access control. Ouaddah et al [89] have extensively investigated access control in IoT environments. He et al [59] have recently proposed a new perspective of access control policies specifications for home IoT. However, few solutions in the literature are proposed specifically to meet smart home IoT challenges

The rest of this section provides an analysis of IoT access control models from the literature. The models are categorized according to their foundational model, viz., RBAC, ABAC, UCON and CapBAC.

2.2.1 IoT Access Control Models Based On RBAC

Covington et al [36] developed GRBAC for aware homes which introduced the notion of environment and device roles, to capture environmental conditions and to enable devices categorization respectively, but did not give a formal model. Subsequently they provided a brief but incomplete

formalization without implementation [35]. In [122] the authors extended RBAC by introducing context constraints. However, they mainly focused on the environment of web services. Researchers in [25, 67] proposed two different solutions, but both of them are focused on Web of Things [38, 110]. Their models are not adequate for smart homes. In the first solution, the architecture is completely centralized in a central access control decision facility coupled to a database, whereby access control decisions are taken outside the house requiring a live connection and increasing the attack surface. On the other hand, the main drawback of the second solution is the strong attachment to Social Network Services (SNS). Resource owners and requesters must have an SNS profile or account to interact with each other which is unsuitable in case of smart homes where we have kids that may not have a social network account, and we may have workers with whom one may not want to connect in social networks, like a plumber who should access the house for one time. Moreover, this solution introduces the SNS provider as a trusted third party. In [68] an attribute-role based hybrid access control model was introduced for IoT in general and not specific to smart homes. Also, no implementation was provided. The RBAC model for IoT was also adopted in [74]. However, the authors focus on providing an authentication protocol, while they only gave a high level overview of their RBAC model.

2.2.2 IoT Access Control Models Based on ABAC

In [26], the authors introduced an ABAC-based model that focuses on device to device access control in smart homes. However, use cases and performance evaluation are lacking. Other access control models that are based on ABAC for IoT were proposed in [23, 81, 119, 121]. However, as observed by [17], it is not simple to design and implement an adequate ABAC model for IoT given that the implementation of ABAC usually requires heavy computation, which cannot be supported by constrained smart things. Furthermore, increasing the number of attributes can significantly increase the chance of conflict among the access policies, and it is not easy to detect and resolve these conflicts. Finally, identifying a set of sufficient attributes is critical, but also challenging. We should mention here that in [81], and [119] 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. Moreover, an ABE model for smart grids may not be suitable for computationally constrained smart home things. In [56] the authors introduced a formalized dynamic, and fine grained ABAC model for smart cars, which is not applicable to the smart home case. Recently, Bhatt et al [28] proposed a conceptual attribute based access control and communication control model for IoT. However, their access control model doesn't capture environment attributes.

2.2.3 IoT Access Control Models Based on CapBAC

Much work has been done in the literature using CapBAC in IoT. The major drawback in CapBAC model is that it requires that all devices must implement CapBAC, which is unlikely given the heterogeneity of a home smart things. Moreover, in CapBAC individual devices or gateways should act as policy decision points, which can be inconvenient on computationally and power constrained devices. Authors in [89] gives a survey on solutions proposed using CapBAC model.

2.2.4 IoT Access Control Models Based on UCON

The distinguishing properties of usage control (UCON) beyond traditional ABAC are the continuity of access decisions and the mutability of subject and object attributes [90, 92, 123]. A few solutions have been proposed in the literature that are based on UCON. However, these models cannot be adopted yet for various reasons. In [54], the model is proposed as a Device to Services (D-S) access control model, moreover, no implementation was provided, instead, only two theoretical experiments were introduced and assessed. In [71], the authors mainly focused on providing a distributed Peer-to-Peer (P2P) architecture. They did not consider how to use their system to grant users access to different smart things in the house. Finally in [77], the authors did not consider justifying and illustrating the fitness of their model for smart home IoT access control challenges.

2.2.5 IoT Access Control Models Based on Blockchains

Some solutions built on blockchain technology have been proposed (e.g. [41, 84, 88]. However, as [84] described, the blockchain technology has some technical characteristics that could limit its applicability. First, cryptocurrency fees are typically a fundamental part of blockchain-based platforms. All the transactions include a fee. Second, processing time, transactions take time to get accepted into the blockchain.

2.2.6 IoT Access Control Models Based on Other Models

In the literature, several other access control models for IoT have been proposed. The authors in [17, 89, 94, 97, 125] provide surveys on these. However, none of them provided an access control model that meet smart home IoT challenges and at the same time formalized, justified and implemented.

CHAPTER 3: ROLE-BASED ACCESS CONTROL MODEL FOR SMART HOME IOT AND RELATED CRITERIA (EGRBAC)

In this chapter, we propose a criteria for smart home IoT access control models. Moreover, we introduce the extended generalized role based access control (EGRBAC) model for smart home IoT. EGRBAC is a dynamic and fine-grained model, suitable for constrained home environments. We provide a formal definition of the model and illustrate its features by a use case scenario. We further provide an analysis of the beneficial attributes of EGRBAC as well as its limitations. Finally, we provide a proof-of-concept implementation in Amazon Web Services (AWS) IoT platform, followed by discussion of future enhancements. Significant portion of this chapter build on the following publication [19] with some revisions and updates.

S. Ameer, J. Benson and R. Sandhu, "The EGRBAC Model for Smart Home IoT," 2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), 2020, pp. 457-462.

3.1 Criteria for Smart Home IoT Access Control Models

He et al [59] have recently proposed a new perspective of access control policies specifications for home IoT. They identify four access control policy characteristics that need to be maintained in smart homes, as follows. (i) Access control should be fine-grained at the level of individual operations on devices (called capabilities in [59]) rather than at the device level. (ii) The complex social relationships between the house members play an important role in access control policies. (iii) Smart home IoT access control policies are highly impacted by contextual factors. (iv) There are some commonly occurring preferences amongst home users that can be configured as a default setting.

Here, we introduce our specifications for smart home IoT access control models. The first two characteristics are inspired by He et al [59] perspective. We believe that a smart home IoT access control model (whether it is device to device (D-D), user to device (U-D) or both) should exhibit,

at least, the following characteristics.

1. The model should be dynamic so as to capture environment and object contextual information.

2. The model should be fine-grained so that a subset of the functionality of a device can be authorized rather than all-or-nothing access to the device.

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

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

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

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

We analyzed IoT access control models proposed in the literature based on these six characteristics. A summary of the analysis is provided in Table 3.1. In this table we only included access control models that govern user to device access, since this is the scope of our research. The IoT access control models in Table 3.1 are based on RBAC [48, 69, 100], ABAC [63, 66], UCON [93] or CapBAC [89]. Going beyond the models in this table, some approaches based on blockchain technology have been proposed (e.g. [41, 84, 88]. However, as [84] described, the blockchain technology has some technical characteristics that could limit its applicability such as, cryptocurrency fees and slow processing time. The authors in [17, 89, 125] provide surveys on additional models

Table 3.1: Characteristics of IoT Access Control Models

Model Type	Model	U-D or D-D	Dynamic	Fine Grained	Suitable for constrained home environment	Designed or interpreted for smart home IoT	Implemented	Provides a formal Access Control Model
RBAC Model	Covington et al [36]	U-D	yes	no	yes	yes	no	no
RBAC Model	Zhang et al [122]	U-D and D-D	yes	yes	yes	no	no	yes
RBAC Model	Barka et al [25]	U-D and D-D	no	yes	no	no	no	utilizes RBAC [101]
RBAC Model	Jindou et al [67]	U-D	no	yes	no	no	yes	yes
RBAC Model	Kaiwen et al [68]	U-D	yes	yes	yes	no	no	yes
RBAC Model	Liu et al [74]	U-D	no	yes	yes	no	no	no
ABAC Model	Ye et al [121]	U-D and D-D	yes	no	no	no	no	yes
ABAC Model	Bandara et al [23]	U-D	no	yes	yes	no	yes	utilizes XACML [98]
ABAC Model	Mutsvangwa et al [81]	U-D	N/A	N/A	no	no	no	no
ABAC Model	Xie et al [119]	U-D and D-D	N/A	N/A	no	no	no	no
UCON Model	Martinelli et al [77]	U-D	yes	yes	yes	no	yes	utilizes U-XACML [31, 72]
CapBAC Model	A survey is provided in [89]	Not adequate for the constrained environment of smart homes as explained in Section 2.2.3.						

beyond Table 3.1 such as focused on D-D only. From the table we can notice that, no model satisfies all desired characteristics. Furthermore, surprisingly, except for [36], no model was designed or interpreted explicitly for smart home environment.

3.2 EGRBAC Model for Smart Home IoT

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

3.2.1 Motivation

Smart homes have been a vision of future society. IoT plays an important role to make home smarter by introducing smart devices and sensors that are capable of interacting together and to end users.

Providing an appropriate access control model for IoT services is a vital but challenging topic. The deployment of resource constrained devices along with the adoption of a plethora of technologies enlarges the attack surface and introduces new security vulnerabilities [89, 97]. The shortcoming in applying access control policies in IoT applications leads to devices and apps being easily exploited to gain unauthorized access to devices and to users and devices data [44, 59, 89]. Real world examples of the shortcomings of current access control policy specification and authentica-

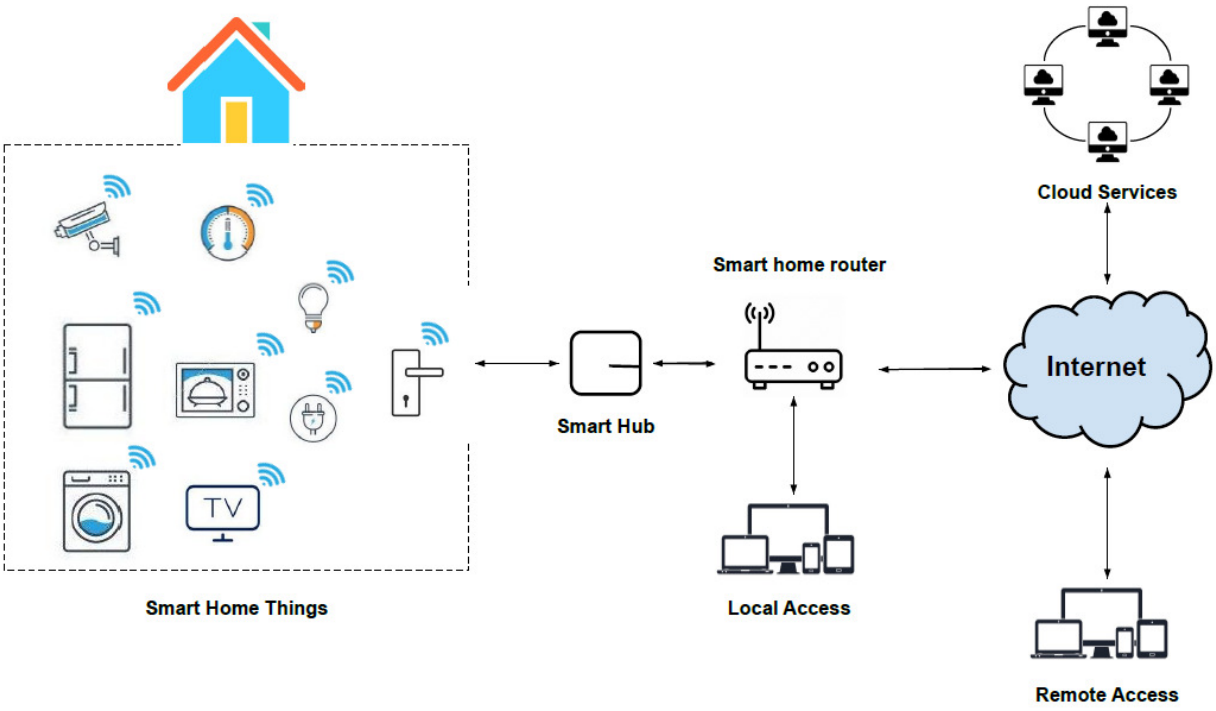

Figure 3.1: EGRBAC Architecture (adapted from [51])

tion for home IoT devices have begun to appear as described in [59], [114], and [61]. Security and privacy in IoT are primary factors that will enable wide adoption of IoT especially at the consumer level. It is generally accepted by most researchers that access control is a critical service in IoT.

To date, IoT security and privacy research has focused on such devices' insecure software-engineering practices, improper information flows, and the inherent difficulties of patching networked devices. Surprisingly little attention has been paid to access control policy specification, or authentication in home IoT [59]. Authorization issues have been explored extensively for many different domains. However, as discussed in Section 1.1, the characteristics that make IoT distinct from prior computing domains necessitate a rethinking of access control. In particular, the need arises for a dynamic and fine-grained access control mechanism, where users and resources are constrained [89].

3.2.2 Background

In this section we introduce GRBAC (Generalized Role Based Access Control) model [36]. EGRBAC is in part inspired by this model. We also present an IoT based smart home architecture [51], which we adopted to enforce EGRBAC.

The GRBAC Model

Covington et al introduced the Generalized Role-Based Access Control (GRBAC) model [36]. In addition to the usual concept of *User Roles*, GRBAC incorporates the notion of *Object Roles* and *Environment Roles*. A user role is analogous to the traditional RBAC role. An object role is defined as the properties of the resources in the system, such as images, source code, streaming videos, devices. An environment role is defined as the environment state during access. Covington et al [35] subsequently described an architecture to support environment roles activation according to the current environment conditions. They also provided a high level but incomplete formal definition of environment role based access control model, building upon [101]. They did not consider formalizing the object role part of GRBAC. In this paper, we provide a more fine grained model with a detailed formalization. However, we used devices instead of objects since it is more appropriate for smart homes.

IOT Based Smart Home Architecture

The smart home IoT architecture that we adopted for EGRBAC enforcement was introduced by Geneiatakis et al [51]. It is illustrated in Figure 3.1. The IoT devices are connected to a corresponding hub and are not directly accessed by other devices or by users. The intermediate hub is responsible for providing Internet connectivity, since the majority of commercial sensors do not provide direct Internet connectivity. The communication between the smart hub and the IoT devices is usually wireless, through different protocols such as Zigbee, Z-Wave and WiFi. In order to connect the smart IoT devices optionally, to the outside world, the hub is connected to the home's routers via an Ethernet or a Wi-Fi interface. In general there are two types of access. In local

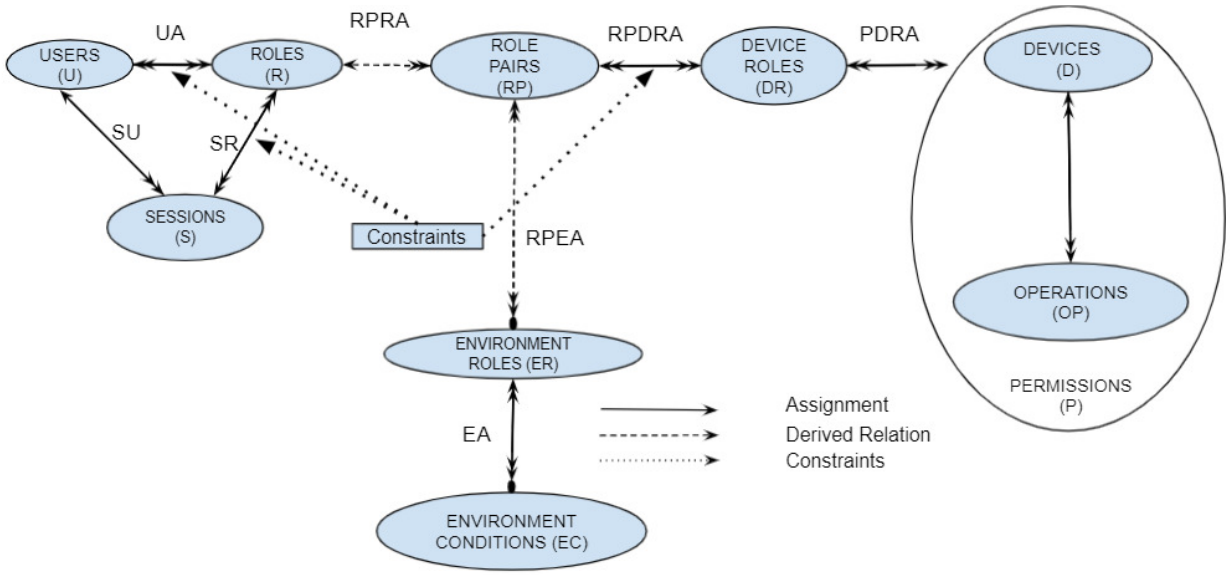

Figure 3.2: EGRBAC Model Components

access users directly interact with the IoT devices through the connectivity services provided by the hub. In remote access users access IoT devices via cloud services, which in turn communicate with the smart hub via the Internet to access these devices.

3.2.3 EGRBAC Formal Definition

Figure 3.2 depicts the components of EGRBAC, and Table 3.2 formally defines these. Sets are shown as ovals in Figure 3.2, while the binary relations amongst them are shown as directed arrows with the single arrow indicating "one" and the double arrow "many." An arrow ending in a dot indicates a subset rather a single element of that set (as in one end of EA and $RPEA$). A solid arrow represents assignment, a dashed arrow indicates a derived relation via mathematical definitions, for example, $RPRA$ and $RPEA$ relations are determined by definition from RP and hence are derived relations rather than independent assignments. Dotted arrows represent constraints.

Users (U), Roles (R), and Sessions (S) are familiar sets in RBAC systems. A user is a human being who interacts with smart home devices as authorized. In context of smart homes, a role specifically represents the relationship between the user and the family, which encompasses parents, kids, neighbors and such [59]. The many-to-many UA relation specifies the assignment

Table 3.2: $EGRBAC$ Model Formalization

Users, Roles and Sessions

$-U$ and R are sets of users and roles respectively (home owner specified)

$-UA \subseteq U \times R$, many to many user role assignment relation (home owner specified)

We define the derived function $roles(u) : U \rightarrow 2^R$, where: $roles(u_i) = \{r_j \mid (u_i, r_j) \in UA\}$

$-S$ is the set of sessions (each session is created, terminated and controlled by an individual user)

$-SU \subseteq S \times U$, many to one relation assigning each session to its single controlling user

We define the derived function $user(s) : S \rightarrow U$, where: $user(s_i) = u_j$ such that $(s_i, u_j) \in SU$

$-SR \subseteq S \times R$, many to many relation that assigns each session to a set of roles that can be changed by the controlling user

We define the derived function $roles(s) : S \rightarrow 2^R$, where: $roles(s_i) = \{r_j \mid (s_i, r_j) \in SR\}$

It is required that $roles(s) \subseteq roles(user(s))$ at all times

Devices, Operations, Permissions and Device Roles

$-D$ is the set of devices deployed in the smart home (home owner deployed)

$-OP$ and $P \subseteq D \times OP$ are sets of operations and permissions respectively (device manufacturers specified)

$-DR$ is the set of device roles (home owner specified)

$-PDRA \subseteq P \times DR$, many to many permissions to device roles assignment (home owner specified)

We define the derived function $droles(p) : P \rightarrow 2^{DR}$, where: $droles(p_i) = \{dr_j \mid (p_i, dr_j) \in PDRA\}$

Environment Conditions and Environment Roles

$-EC$ is the set of boolean environment conditions (determined by sensors deployed in the smart home under home owner control)

At any moment each $ec_i \in EC$ is either True or False depending on the state of the corresponding sensor

$-ER$ is the set of environment roles (home owner specified)

$-EA \subseteq 2^{EC} \times ER$, many to many environment role activation relation (home owner specified)

At any moment, $er \in ER$ is activated iff $(\exists(ec_{i1}, ec_{i2}, \ldots, ec_{in}), er) \in EA)[ec_{i1} \wedge ec_{i2} \wedge \ldots \wedge ec_{in} = \text{True}]$ at that moment

Role Pairs

$-RP \subseteq R \times 2^{ER}$, many to many role pairings of user role and subsets of environment roles (home owner specified)

For $rp = (r_i, ER_j) \in RP$, we define $rp.r = r_i$ and $rp.ER = ER_j$

We define the derived relation $RPRA \subseteq RP \times R$ where: $RPRA = \{(rp_m, r_n) \mid rp_m \in RP \wedge rp_m.r = r_n\}$

We define the derived relation $RPEA \subseteq RP \times 2^{ER}$ where: $RPEA = \{(rp_m, ER_n) \mid rp_m \in RP \wedge ER_n = rp_m.ER\}$

Role Pair Assignment

$-RPDRA \subseteq RP \times DR$, many to many RP to DR assignment (home owner specified)

Constraints

$-PRConstraints \subseteq 2^P \times 2^R$, many to many permission-role constraints relation (home owner specified)

For each $(P_i, R_j) \in PRConstraints$ it is required that

$(\forall p_m \in P_i)(\forall r_n \in R_j)(\forall (rp_p, dr_q) \in RPDRA)[(p_m, dr_q) \notin PDRA \vee rp_p.r \neq r_n]$

$-SSDConstraints \subseteq R \times 2^R$, many to many static separation of duty constraints relation (home owner specified)

For each $(r_i, R_j) \in SSDConstraints$ it is required that $(\forall u \in U)(\forall r \in R_j)[(u, r) \in UA \implies (u, r_i) \notin UA]$

$-DSDConstraints \subseteq R \times 2^R$, many to many dynamic separation of duty constraints relation (home owner specified)

For each $(r_i, R_j) \in DSDConstraints$ it is required that $(\forall s \in S)(\forall r \in R_j)[(s, r) \in SR \implies (s, r_i) \notin SR]$

CheckAccess Predicate

$-CheckAccess$ is evaluated when session s_i attempts operation op_k on device d_j while the environment conditions in EC_l are True

$-CheckAccess(s_i, op_k, d_j, EC_l)$ evaluates to True or False using the following formula:

$$(\exists(rp_m, dr_n) \in RPDRA)\ [((d_j, op_k), dr_n) \in PDRA \wedge (s_i, rp_m.r) \in SR \wedge rp_m.ER \subseteq \{er \in ER \mid (\exists EC'_l \subseteq EC_l)[(EC'_l, er) \in EA]\}]$$

of users to roles. An example of a user with two different roles is a neighbor who is assigned the neighbor role, but also happens to be a plumber who needs temporary access to repair an appliance and so should have different set of privileges for that purpose in a worker role. Users establish sessions during which they may activate a subset of the roles they are assigned to. A user might have multiple sessions active simultaneously. SU is a many to one relation that maps each session to its unique controlling user. SR is a many to many relations that maps each session to the set of roles associated with it. The function $roles$ maps each user to the set of roles assigned to it. The function $user$ maps each session to its unique user creator.

A Device (D) is a smart home device such as a smart TV. Operations (OP) represent actions on devices as specified by device manufacturers. A permission is an approval to perform an operation on one device, i.e. it is a device, operation pair. The set of permissions P is a subset of $D \times OP$. In EGRBAC, Device Roles (DR) are means of categorizing permissions of different devices (different from GRBAC where Device Roles categorize devices including all their permissions). For example, we can categorize the dangerous permissions of various smart devices by creating a device role called dangerous devices and 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 $PDRA$ relation specifies this assignment. The function $droles$ maps each permission to the set of device roles assigned to it.

Environment Roles (ER) are a GRBAC innovation representing environmental contexts, such as daytime/nighttime, and winter/summer. Environment roles are turned on/off (i.e., triggered) by Environment Conditions (EC) such as daylight, or weather. The environment activation assignment relation EA maps each environment role to a subset of EC. Suppose $Entertainment\_Time$ should be active on weekend evenings. We can use $weekends$, active during weekends, and $evenings$, active during evenings, and assign ($\{weekends, evenings\}$, $Entertainment\_Time$) to EA. Each role pair is a combination of a role and currently active 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 $rp.ER$ that is the subset of environment roles associated with rp. The permissible role pairs RP

are specified as a subset of $R \times 2^{ER}$, since some ER subsets may not be meaningful. The derived relation $RPRA$ associates each role to one or more role pairs. Similarly, $RPEA$ associates each role pair to a subset of ER. $RPDRA$ brings all these components together by assigning device roles to role pairs, and hence, for each role pair rp, the single role associated to it through $RPRA$ can get access to all device roles assigned to it through $RPDRA$, when the set of environment roles which are associated to rp through $RPEA$ are active.

The main idea in EGRBAC as a whole is that a user is assigned a subset of roles and, according to the current active roles in one of his sessions and the current active environment roles, some role pairs will be active, whereby in that same session the user will get access to the permissions assigned to the device roles which are assigned to the current active role pairs.

Table 3.2 also formalizes the component Constraints which is discussed in Section 3.2.5.

The bottom part of Table 3.2 formalizes the authorization check access predicate of EGRBAC. Consider a session s_i which attempts to perform an operation op_k on a device d_j when the subset of environment conditions EC_l are active. This operation will succeed if and only if there is a role pair rp_m and a device role dr_n assigned to each other in $RPDRA$ such that the following conditions are true. (i) dr_n is assigned to the permission (d_j, op_k) in $PDRA$. (ii) $rp_m.r$ is one of the active roles of s_i (as given in SR). (iii) Each environment role $er \in rp_m.ER$ is active because it is activated by a subset of the currently active environment conditions EC_l.

3.2.4 EGRBAC Use Case

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

EGRBAC can be configured as shown in Figure 3.3 to achieve these objectives. The five users

$U = \{alex, bob, susan, james, julia\}$
$R = \{kids, parents, babySitters, guests, neighbors\}$
$UA = \{(alex, kids), (bob, parents), (susan, babySitters),$
$(james, guests), (julia, neighbors)\}$

$D = \{TV, DVD, PlayStation, FrontDoorLock, Oven\}$
$OP = OP_{TV} \cup OP_{DVD} \cup OP_{PlayStation} \cup OP_{FrontDoorLock} \cup OP_{Oven}$, where
$OP_{TV} = \{On_{TV}, \mathit{Off}_{TV}, G_{TV}, PG_{TV}, R_{TV}\}$,
$OP_{DVD} = \{On_{DVD}, \mathit{Off}_{DVD}, G_{DVD}, PG_{DVD}, R_{DVD}\}$,
$OP_{PlayStation} = \{On_{PlayStation}, \mathit{Off}_{PlayStation}, G_{PlayStation}, PG_{PlayStation}, R_{PlayStation}\}$,
$OP_{FrontDoorLock} = \{Lock_{FrontDoorLock}, Unlock_{FrontDoorLock}\}$,
$OP_{Oven} = \{On_{Oven}, \mathit{Off}_{Oven}\}$
$P = P_{TV} \cup P_{DVD} \cup P_{PlayStation} \cup P_{FrontDoorLock} \cup P_{Oven}$, where
$P_{TV} = \{TV\} \times OP_{TV}$,
$P_{DVD} = \{DVD\} \times OP_{DVD}$,
$P_{PlayStation} = \{PlayStation\} \times OP_{PlayStation}$,
$P_{FrontDoorLock} = \{FrontDoorLock\} \times OP_{FrontDoorLock}$,
$P_{Oven} = \{Oven\} \times OP_{Oven}$
Let $P_1 = \{TV\} \times \{On_{TV}, \mathit{Off}_{TV}, G_{TV}\} \cup \{DVD\} \times \{On_{DVD}, \mathit{Off}_{DVD}, G_{DVD}\} \cup$
$\{PlayStation\} \times \{On_{PlayStation}, \mathit{Off}_{PlayStation}, G_{PlayStation}\}$

$DR = \{Dangerous\_Devices, Entertainment\_Devices, Kids\_Friendly\_Content\}$
$PDRA = (P_{TV} \cup P_{DVD} \cup P_{PlayStation}) \times \{Entertainment\_Devices\} \cup$
$P_1 \times \{Kids\_Friendly\_Content\} \cup$
$(P_{Oven} \cup P_{FrontDoorLock}) \times \{Dangerous\_Devices\}$

$EC = \{weekends, evenings, TRUE\}$
$ER = \{Entertainment\_Time, Any\_Time\}$
$EA = \{(\{weekends, evenings\}, Entertainment\_Time), (TRUE, Any\_Time)\}$

$RP = \{(kids, \{Entertainment\_Time\}), (parents, \{Any\_Time\}),$
$(babySitters, \{Any\_Time\}), (guests, \{Any\_Time\}),$
$(neighbors, \{Any\_Time\})\}$

$RPDRA = \{((parents, \{Any\_Time\}), Dangerous\_Devices),$
$((kids, \{Entertainment\_Time\}), Kids\_Friendly\_Contents),$
$((parents, \{Any\_Time\}), Entertainment\_Devices),$
$((babySitters, \{Any\_Time\}), Entertainment\_Devices),$
$((guests, \{Any\_Time\}), Entertainment\_Devices),$
$((neighbors, \{Any\_Time\}), Entertainment\_Devices)\}$

Figure 3.3: EGRBAC Use Case Configuration

$alex$, bob, $susan$, $james$, and $julia$, are respectively assigned to roles $kids$, $parents$, $babysitters$, $guests$ and $neighbors$. The devices comprise TV, DVD, $PlayStation$, $FrontDoorLock$, and $Oven$. Each device has different permissions as indicated in P_{TV}, P_{DVD}, $P_{PlayStation}$, $P_{FrontDoorLock}$ and P_{Oven}. Also P_1 is a subset of $P_{TV} \cup P_{DVD} \cup P_{PlayStation}$.

We have three device roles with $PDRA$ assigning P_{TV}, P_{DVD}, and $P_{PlayStation}$ permissions to $Entertainment\_Devices$, P_1 to $Kids\_Friendly\_Content$, and P_{Oven} and $P_{FrontDoorLock}$ to $Dangerous\_Devices$.

Three environment conditions, $weekends$, $evenings$ and $TRUE$ are defined to be respectively active on weekends, evenings and always. EA specifies that the environment role $Entertainment\_Time$ is active when both environment conditions $weekends$ and $evenings$ are active while Any_Time is always active.

$RPDRA$ has the following assignments. The role pair $(parents, \{Any\_Time\})$ is assigned to the device role $Dangerous\_Devices$, whereby parents can use permissions P_{Oven} and $P_{FrontDoorLock}$ without environmental restrictions. The role pair $(kids, \{Entertainment\_Time\})$ is assigned to the device role $Kids\_Friendly\_Content$, so that kids are restricted to P_1 permissions and only when the environment role $Entertainment\_Time$ is active. The role pairs $(parents, \{Any\_Time\})$, $(babySitters, \{Any\_Time\})$, $(guests, \{Any\_Time\})$, $(neighbors, \{Any\_Time\})$ are assigned to the device role $Entertainment\_Devices$ so that users with these roles can use all permissions on $Entertainment\_Devices$ at any time.

3.2.5 EGRBAC Constraints

An important component in EGRBAC is Constraints. A constraint is an invariant that must be maintained at all times. Constraints are an integral part of RBAC and ABAC models [29, 48, 101]. In EGRBAC, we define three types of constraints, as follows.

Permission-role constraint. These constraints prevent specific roles from getting access to specific permissions. In the use case above, the permissions embodied in the $Dangerous\_Devices$ role are assigned to the $(parents, \{Any\_Time\})$ role pair in $RPDRA$. However, this does not prevent as-

signment of $Dangerous\_Devices$ to other role pairs, perhaps even to $(kids, \{Any\_Time\})$. The latter assignment could happen inadvertently or maliciously. Permission-role constraints prevent such situations.

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

$$(\forall (rp_p, dr_q) \in RPDRA)$$
$$[(p_m, dr_q) \notin PDRA \vee rp_p.r \neq r_n]$$

Thus, it is forbidden to assign any device role that p_m is assigned to, to any role pair with r_n as the role part. EGRBAC use case shown in Figure 3.3 can be augmented with the constraint shown below.

$$PRConstraints = \{(P_{FrontDoorLock} \cup P_{Oven}), R \setminus \{parents\})\}$$

This will prevent the assignment of any permissions in $P_{FrontDoorLock}$ or P_{Oven} to role pairs with the role part being any role except for $parents$.

Static Separation of Duty (SSD). This is the familiar SSD in RBAC. It enforces constraints on the assignment of users to roles. In other words, if a user is authorized as a member of one role, the user is prohibited from being a member of a second conflicting role [99].

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

$$(\forall u \in U)(\forall r \in R_j)[(u, r) \in UA \Rightarrow (u, r_i) \notin UA]$$

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

Dynamic Separation of Duty (DSD). This is the familiar DSD in RBAC. With DSD it is permissible for a user to be authorized as a member of a set of roles which do not constitute a conflict

of interest when acted in independently, but produce policy concerns when allowed to be acted simultaneously [99] in the same session.

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

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

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

3.2.6 EGRBAC Analysis and Limitations

Model Analysis

In the following we analyze EGRBAC against our criteria for smart home IoT which proposed in Section 3.1. According to our criteria, a smart home IoT access control model (whether it is device to device (D-D), user to device (U-D) or both) should be fine grained, dynamic, suitable for constrained home environment, designed and interpreted for smart home IoT, Implemented and tested, and formally defined.

Dynamic We consider our model as a dynamic model. Environment conditions and environment roles allow us to give different users access rights to different capabilities under specific environment contextual factors. Moreover, device roles enable our model to give users access to some permissions, or to some devices based on different contextual characteristics.

Fine grained As illustrated in EGRBAC use case Figure 3.3, our model is able to give users access to some permissions within a single device without the need to give them the access to the entire device, which makes it a capability centric model instead of a device centric model.

Suitable for constrained home environment Our model is suitable for constrained smart home environment for two main reasons: (a) It is built on top of RBAC model, which is considered as a simple model. EGRBAC exploits the organizational power of roles for grouping environment states and objects, in addition to subjects. It authorizes or revokes access based on roles instead

of on an individual basis. Establishing a set of roles in a small or medium-sized environment such as houses is not a challenging task. (b) The enforcement architecture that we adopt (see Section 3.2.2) includes the component smart hub, which facilitates transferring the policy decision engine to a trusted local device. This enables devices to collect and analyze data externally, but closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. Having such setting allows smart homes to enforce EGRBAC model without the need to incorporate advanced or computationally intensive smart things. Moreover, mediating each request through smart hub instead of directly accessing the smart devices solves the heterogeneity problem of IoT devices.

Designed and interpreted for smart home IoT Our model is developed to fit smart home IoT access control challenges. This model is dynamic, fine grained , and captures the complex relationships between home users.

Implemented and tested Our model is demonstrated with an illustrative use case, an AWS implementation that captures local, and remote access for smart home devices as described in Section 3.2.7, and a performance analysis.

U-D or D-D Our model ia a U-D access control model.

Provides a formal Access Control Model Our model is formally structured, and defined as illustrated in Table 3.2.

Policy Conflicts

Conflicting policies may occur when you have negative policies, where you prevent specific roles from accessing specific permissions. In EGRBAC model our policies are positive policies where you give roles access to specific permissions by assigning appropriate role pairs to appropriate device roles. Instead of negative policies, EGRBAC uses constraints to prevent a specific role r_n from accessing a specific permission p_m during configuration time. (see Section 3.2.5).

Usability and Expressiveness

One of the important aspects that need to be considered in smart home access control models is usability, since smart home residents are usually constrained, and not willing to deal with complicated systems. We believe that at this point it would be premature to conduct a usability study for EGRBAC. This model is our first step toward building a set of models ranging from relatively simple and complete to incorporating increasingly sophisticated and comprehensive features. It still needs to be further developed and extended. Another important aspect is expressiveness, wether the system is capable of expressing policies that depict users requirements. Our smart home IoT access control model criteria which is introduced in Section 3.1 incorporates the new perspective of access control specifications recently introduced by He et al [59]. He et al validated the expressiveness of their specifications by conducting an online survey-based user study of 425 participants. As we discussed earlier in Section 3.2.6 our model meets our criteria which implies that it meets He et al perspective. However, in order to deploy this model for commercial uses, a more general sophisticated expressiveness study should be conducted.

Limitations

EGRBAC does not handle device to device communication. Furthermore, it doesn't consider continuous verification for access control authorized policies, where the authorization predicate is only examined at the time of request but does not support ongoing controls for relatively long-lived operations or for immediate revocation.

3.2.7 Proof-Of-Concept Implementation

Here, we describe a proof-of-concept implementation of EGRBAC. We simulated the use case provided in Figure 3.3 using AWS (Amazon Web Services) IoT service [4]. The simulation illustrates how the access control model and policies can be configured to establish the applicability of our model utilizing commercially available systems. Moreover, we executed multiple test cases to measure the processing time in different scenarios.

An AWS account is required to configure and deploy the AWS IoT service known as Greengrass. The Greengrass SDK (Software Development Kit) extends cloud capabilities to the edge, which in our case is the smart home. This enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely on local networks [6].

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

To enforce EGRBAC, we utilized two Json files UserRoleAssignment.json and policy.json, where UserRoleAssignment.json defines the assignments of users to their corresponding roles while policy.json defines all other EGRBAC components relevant to the use case. We also utilized the lambda function service in AWS IoT platform [7] to receive the operation requests of users to access the smart devices in the house, analyze each request according to the content of the policy.json and UserRoleAssignment.json files, and finally trigger the desired actions on the corresponding simulated devices. Code was in Python 2.7 and running on a long-lived lambda function with 128 MB Memory Limit, 30 second timeout. The lambda function, the UserRoleAssignment.json file, and the policy.json file are all configured in the Greengrass. We should mention that our system is a default deny system.

Figure 3.4, illustrates how the communication is handled in our implementation when the user tries to send operation request to turn on a smart TV through his mobile phone while he is inside the house. In this case, a request is sent via MQTT protocol to the virtual object (or local shadow) corresponding to his phone in Greengrass. There is a publish/subscribe relation between the user's phone, and the local shadow through the user's private topic *User/Shadow/Update*. The user's

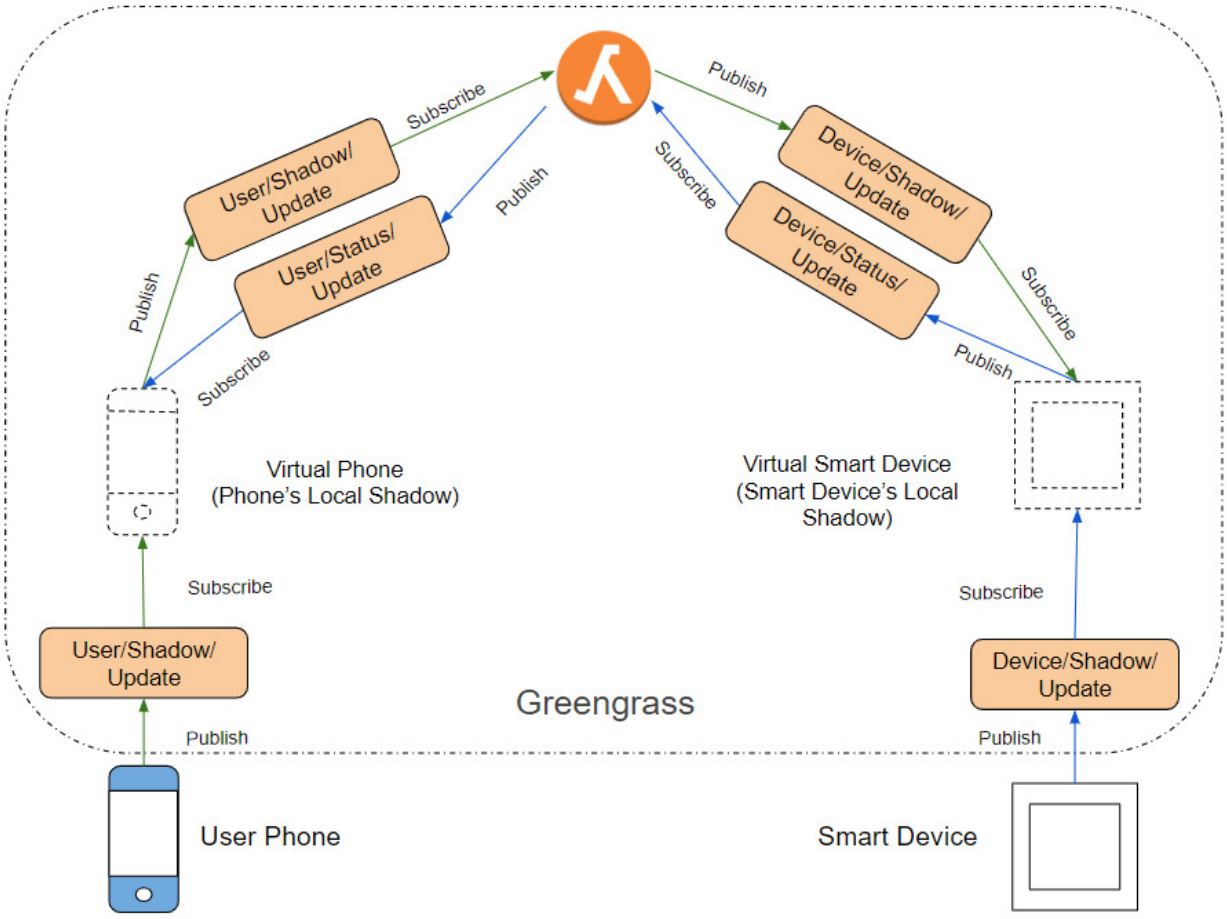

Figure 3.4: Local Request Processing

phone publishes to the topic *User/Shadow/Update*, and the local shadow gets notified with the request. After that, the local shadow publishes to the user's private topic *User/Shadow/Update*, and then since the lambda function is subscribed to this topic it analyzes the request according to the policy.json and UserRoleAssignment.json files and makes a decision wether to allow the user to turn on the TV or not. At this point, there are two cases, either permission is granted or denied. If permission is denied, the lambda function publishes to the user's public topic *User/Status/Update*, the local shadow gets notified and updates the user's phone that the permission was denied. The smart TV in this case does not get an indication that a user attempted to access it. If permission is granted, the smart TV local shadow is notified through the device's private topic *Device/Shadow/Update* and updates the smart TV with the turn on command. After the smart TV is turned on, it publishes to the device's private topic *Device/Shadow/Update* and the TV local

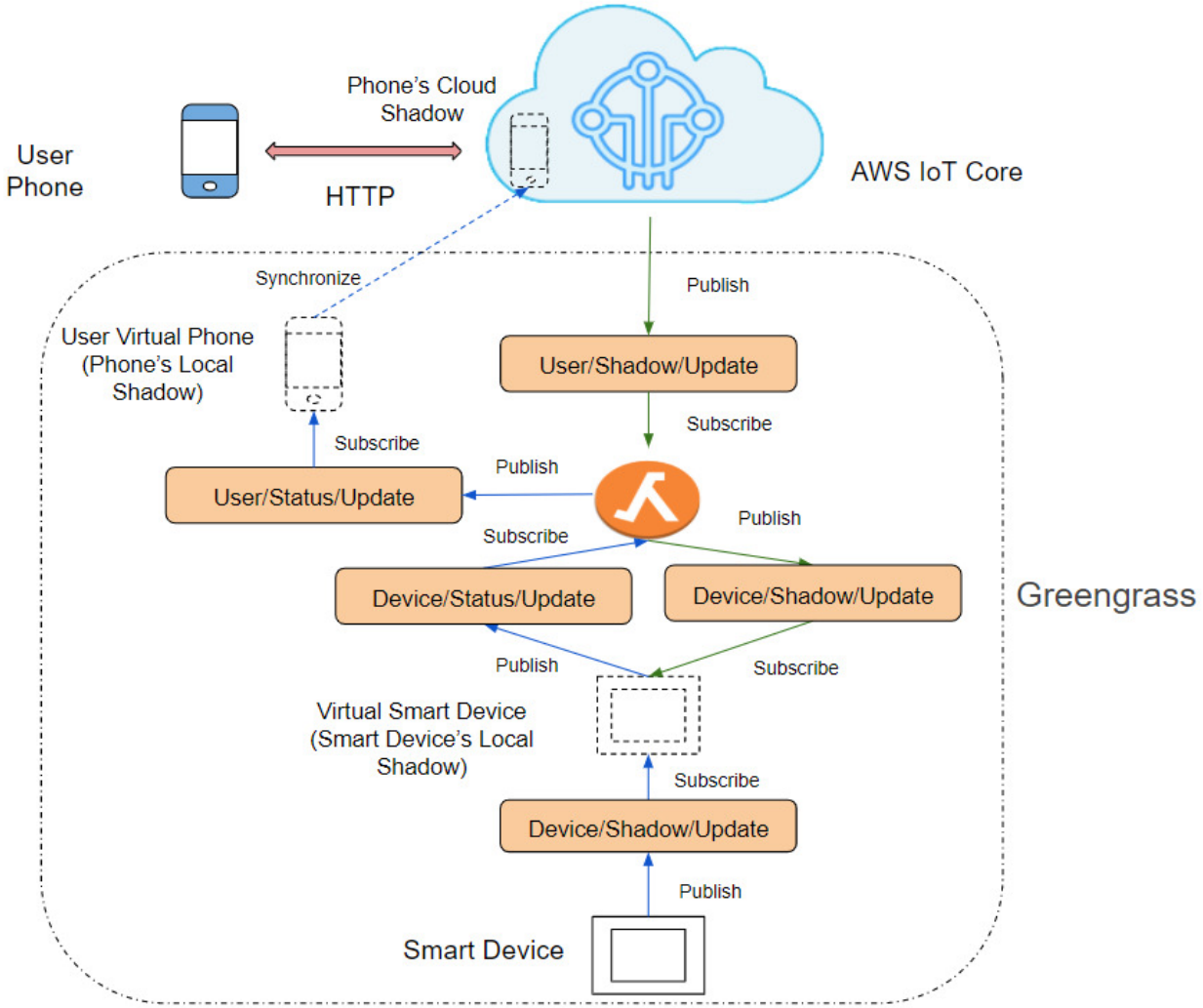

Figure 3.5: Remote Request Handling in Our System

shadow is notified which further notifies the lambda function by publishing to the device's public topic *Device/Status/Update*. The lambda function then notifies the user phone's local shadow which in turn updates the user's phone that the TV was turned on successfully.

Figure 3.5, illustrates how the communication is handled in our implementation in case of remote access. If a user Bob is trying to turn on the oven using his smart phone from a remote place. First, a request is sent through the HTTP send protocol to the cloud's synchronized shadow state of the user device, in this case the user's phone. Once the user's phone state is changed on the cloud, the cloud forwards the message to the local Greengrass lambda by publishing to the user's private topic *User/Shadow/Update*, the lambda receives the request, analyzes it according

to the access control policies defined in the policy.json file and the UserRoleAssignment.json file and makes a decision to allow the user to turn on the oven or no. If the access is granted, the lambda function will send the request to the smart device Greengrass's local shadow by publishing to the device's private topic *Device/Shadow/Update*, the local shadow will get the request and will automatically update the smart device (smart oven in this case) to turn on. When the smart device perform the operation, it notifies its local shadow by publishing to the device's private topic *Device/Shadow/Update*, the local shadow then notifies the lambda by publishing to the device's public topic *Device/Status/Update*, the lambda then updates the user's phone local shadow by publishing to the user's public topic *User/Status/Update*. The user's phone local shadow automatically synchronizes this state to the cloud shadow which in turn notifies the user's phone that the request has been served. On the other hand, if the decision was not to allow this operation to be performed, the lambda function would publish to the user's public topic *User/Status/Update*, the local shadow would get notified and would automatically synchronize this state to the cloud's synchronized shadow state of the device. The cloud's shadow would then update the user's phone through the http send protocol that the permission was denied and the user has no right to turn on the oven. The smart oven, in this case, would never get an indication that a user attempted to access it.

Performance results

We executed multiple test cases to measure the processing time in different scenarios. In our performance testing, we implemented the configuration of Figure 3.3. In the following test cases, we measure the average lambda function execution time under different conditions. Table 3.3 shows the average lambda function execution time when we send multiple requests from one user to multiple devices. The first, second, and third rows show the average time when the parent Bob requests to unlock the door lock, the average time when Bob requests to turn on the oven, the TV, and the DVD at the same time, and the average time when Bob requests to unlock the door lock, turn on the oven, the TV, the DVD, and the playStation at the same time respectively. All the requests were approved as they were supposed to according to our configured policies. Table 3.4

Table 3.3: One User Sending Requests to Multiple Devices

Number of Users	Number of devices	Lambda Processing Time in ms.	Total Number of requests
1	1	1.029138	1000
1	3	1.236029	3000 (1000 per request)
1	5	1.202856	5000 (1000 per request)

Table 3.4: 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	1.029138	1000
3	3	1.796938	3000 (1000 per request)
5	5	2.833097	5000 (1000 per request)

Table 3.5: Multiple Users Sending Requests to One Device

Number of Users	Number of devices	Lambda Processing Time in ms.	Total Number of requests
1	1	1.029138	1000
3	1	0.955529	3000 (1000 per request)
5	1	0.956221	5000 (1000 per request)

shows the average lambda function execution time when we send multiple requests from multiple users to multiple devices (one user per device) at the same time. The first, second, and third row show the average time when the parent Bob requests to unlock the door lock, the average time when Bob requests to unlock the door lock, the kid Alex requests to turn on the oven, and the babysitter Susan requests to turn on the TV at the same time, the average time when the three access requests tested in the second row are carried again in addition to, the guest James requests to turn on the DVD, and the neighbor Julia requests to turn on the playStation. The system responded correctly where all the requests were approved except for when the kid Alex requests to turn on the oven. We can conclude that when the number of requests for different users and different devices (one user per device) increases, the lambda processing time also increase. Finally, Table 3.5 shows the average lambda function execution time when we send multiple requests from multiple users to one device at the same time. The first, second, and third rows show the average time when the parent (1 user), the parent, the kid, and the babysitter (3 users), or the parent, the kid, the babysitter, the guest, and the neighbor (5 users) respectively all request to unlock the door at the

same time. The system responded correctly where all the requests were denied except for when the parent Bob requests to unlock the door lock. Here, we can see that the average of the lambda processing time decreases when we have more denies. This result is expected since in order to approve a request, our policy checking engine (the lambda function) implemented to check for the authorization predicate explained in Table 3.2 need to verify each condition in the authorization predicate. On the other hand, if only one of the authorization predicate conditions is violated the lambda function will deny the request without the need to check the rest of the authorization predicate. To conclude, our system takes more time when approving a request than when denying it. Overall, our model is functional, and can be easily applied. Moreover, we can notice that the execution time is generally low.

CHAPTER 4: ATTRIBUTE-BASED ACCESS CONTROL MODEL FOR SMART HOME IOT (HABAC)

In this chapter, we introduce the smart home IoT attribute-based access control model (HABAC). HABAC is a dynamic and fine-grained model that is developed specifically to meet smart home IoT challenges. This chapter provides an analysis of HABAC relative to the previously introduced EGRBAC (extended generalized role based access control) model. It compares the theoretical expressive power of these models by providing algorithms for converting an HABAC specification to EGRBAC and vice versa, and discuss the insights for practical deployment of these models resulting from these constructions. Major sections of this chapter are based on the following publication [20] with some revisions and modifications.

Safwa Ameer, and Ravi Sandhu. "The HABAC Model for Smart Home IoT and Comparison to EGRBAC." Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems. 2021.

4.1 Motivation

In the literature, several access control models have been proposed for IoT in general. The majority of them are built on ABAC or RBAC. Some researchers argue that RBAC is more suitable for IoT since it is simpler in management and review, while ABAC is complex [17, 67, 68]. On the other hand, others argue that ABAC models are more scalable and dynamic, since they can capture different devices and environment contextual information [26, 27, 121]. However, RBAC models can be extended, such as the recent EGRBAC model [19] for smart home IoT which can express environment and device characteristics. RBAC enforcement may also be more lightweight for constrained home environment.

Hence, when it comes to smart homes, at this point it is not fully clear what is the benefit of ABAC over RBAC, and vice versa. Our intuitive insight is that a hybrid model will better capture smart home IoT AC requirements as this was already the case for traditional access control appli-

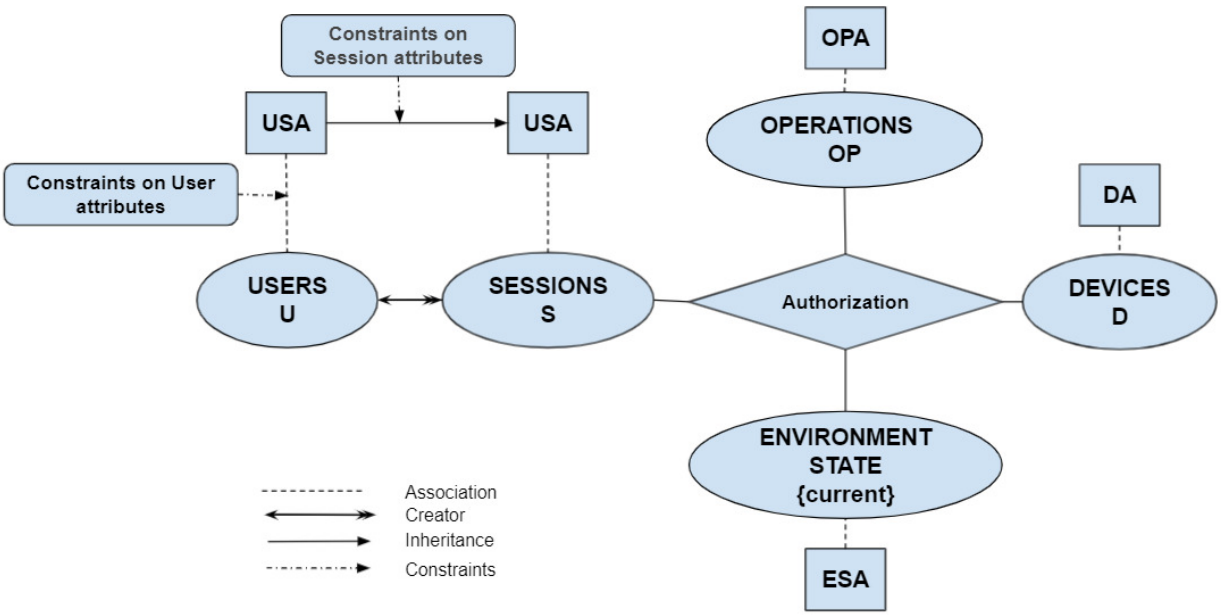

Figure 4.1: HABAC Model

cation domains. In order to further investigate this intuition our approach is to develop pure RBAC and pure ABAC based models explicitly defined to meet smart home challenges, and compare their benefits and drawbacks. This comparison will provide insights to guide us in designing suitable hybrid models.

4.2 HABAC Model for Smart Home IoT

ABAC models utilize attributes of users, sessions (subjects), objects, operations and environment to specify flexible, dynamic, and fine grained authorization policies. These characteristics arguably make ABAC suitable for deployment in complex domains such as smart home IoT. In this section, we introduce our HABAC (Home-IoT Attribute Based Access Control) model developed for user to device interaction in smart home IoT.

4.2.1 HABAC Formal Definition

The HABAC model is inspired by the ABAC model of Xin et al [66], extended to include environment attributes. Figure 4.1 depicts HABAC components. Users (U), Operations (OP), Devices (D), and Environment States (ES) are sets and shown in ovals. User/Session Attributes (USA),

Table 4.1: $HABAC$ Model Formalization Part I: Basic Sets and Components

Basic Sets and Functions

$-U$ is a finite sets of users (home owner specified)

$-S$ is the set of sessions (each session is created, terminated and controlled by an individual user)

$-$The function $user(s) : S \rightarrow U$ maps each session to its unique creator and controlling user

$-D$ is the set of devices deployed in the smart home (home owner deployed)

$-OP$ is the set of possible operations on devices (device manufacturers specified)

$-$The function $ops : D \rightarrow 2^{OP}$ specifies the valid operations for each device (device manufacturers 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 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 $att \in USA$, $att(s) = att(user(s))$ at all time
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

$-attType : USA \cup DA \cup OPA \cup ESA \rightarrow \{set, atomic\}$.

$-$Each $att \in USA \cup DA \cup OPA \cup ESA$ 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:

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

$-$Every $att \in USA \cup DA \cup OPA \cup ESA$, att is designated to be either a static or dynamic attribute where dynamic attributes must have corresponding sensors deployed in the smart home (under home owner control)

$-$Static attribute ranges and values are set and changed by administrator actions (by home owner or device manufacturers)

$-$Dynamic attribute ranges and values automatically determined by sensors deployed in the smart home (under home owner control)

Constraints

$-UAConstraint \subseteq UAP \times 2^{UAP}$ is the user attribute constraints relation (home owner specified) where

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

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

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

$-SAConstraint \subseteq UAP \times 2^{UAP}$ is the session attribute constraints relation (home owner 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 4.2 (home owner specified). It is evaluated for a specific session s_i, operation op_k, device d_j and environment state $current$ as specified in Table 4.2

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 4.2: $HABAC$ Model Formalization Part II: Attributes Authorization Function

Attributes Authorization Function

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

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists x \in set.\alpha \mid \forall x \in set.\alpha \mid$

  $set\ setcompare\ set \mid atomic \in set \mid atomic \notin set \mid atomic\ atomiccompare\ atomic$

- $setcompare ::= \subset \mid \subseteq \mid \not\subseteq$
- $atomiccompare ::= < \mid = \mid \leq$
- $set ::= usa(s) \mid opa(op) \mid esa(current) \mid da(d)$, 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

Operation Attributes (OPA), Environment State Attributes (ESA), and Device Attributes (DA) are attribute functions and shown as squares. We have two types of constraints shown in rectangles: constraints on user attributes, and constraints on session attributes. Table 4.1, and Table 4.2 formally define this model.

Basic Sets and Functions:

Users (U) are humans interacting directly with the smart things. Sessions (S) are similar to the concept of subjects in [66], users create sessions during which they may perform some actions in the system, the creating user is the only one who can terminate a session. The function $user(s)$ maps each session s to its unique user creator. Devices (D) are smart home devices such as a smart light. Operations (OP) represent actions on devices as specified by device manufacturers. The function $ops(d)$ maps each device d to the set of valid operations on d. Environment States ($ES = \{current\}$) is a singleton set where $current$ denotes the picture of the environment at the current time instant. This component can be extended in the future to include other instants of time such as yesterday, last week, and so on. However, such generalization requires a careful investigation on how we will track time, and to what extent.

Attribute Functions and Values:

Users, sessions, devices, operations, and environment states have characteristics which are used in

access control decision and expressed as their attributes. An attribute is a function which takes an entity such as a user and returns a specific value from its range. An attribute range is given by a finite set of atomic values. An atomic valued attribute will return one value from the range, while a set valued attribute will return a subset of the range. In general, we have two types of attributes: (a) Static Attributes, this type is relatively static. For example, user relationship, device level of danger, and so on. Range and values of static attributes need administrator action to be set and changed. (b) Dynamic Attributes, this type is rapidly changing due to different conditions. For instance, user location, weather, and device temperature. Ranges and values of dynamic attributes are automatically determined by sensors deployed in the smart home and under home owner control.

User/Session attribute functions (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 $usa(s) = usa(user(s))$. Device attribute functions (DA) is the set of attributes associated with smart things, for instance, "kitchen devices", and "Alex devices". Operation attribute functions (OPA) is the set of attributes assigned to different operations, for example you may want to characterize dangerous operations by creating an operation attribute called "Dangerous Operations" and associate it with those operations. Environment state attribute functions (ESA) is the set of attributes describe the environment condition of the current instance of time. For example, "day", "time", and "weather condition". How different environment state attributes get triggered is outside the scope of our model. Operation, and device attribute functions are partial functions; we may have some devices, or some operations that are not assigned to some attributes. On the other hand, users, sessions, and environment state attributes are total functions.

Constraints:

A constraint is an invariant that must be maintained at all times. In HABAC we define two types of constraints, as follow: (a) Constraints on user attributes: these constraints enforce restrictions on user attributes. For instance, if we have the following user attribute constraint $((Relationship, kid), \{(Adults, True)\})$, this constraint implies that for any user u if $kid \in$

$Relationship(u)$, then it is required that $Adults(u) \neq True$. (b) Constraints on session attributes: these constraints enforce restrictions on session attributes. For instance, if we have the following session attribute constraint: $((Relationship, staying\ Home\ Kid), \{(Relationship, travel\ Abroad\ Kid)\})$, this constraint implies that for any session s if $staying\ Home\ Kid \in Relationship(s)$, then $travel\ Abroad\ Kid \notin Relationship(s)$.

Attributes Authorization Function (Policy):

It is a two-valued boolean function which is evaluated for each access decision. It is defined using the grammer of Table 4.2. 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.

Check Access Predicate:

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

- The operation op_k is assigned to the device d_j by the device manufacturer.
- The authorization function is evaluated to True.

4.2.2 HABAC Use Case

We present a use case to demonstrate the components and configurations of HABAC. The objectives are as follow: (a) Allow kids to use kids friendly operations in entertainment devices (G, PG contents in TV, and $A3$ (games for group age below 3 years old), $A7$ (games for group age below 7 years old) contents in Play Station) during specific time (weekends afternoons and evenings, and weekdays evenings). (b) Authorize teenagers to use dangerous kitchen devices (Oven) only when one of the parents is in the kitchen. (c) Authorize teenagers to use non dangerous kitchen devices

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

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

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

$OP_{TV} = \{G, PG, \ldots\}$
$OP_{PlayStation} = \{A3, A7, A12, BuyGames, \ldots\}$
$OP_{Oven} = \{ON, OFF\}$
$OP_{Fridge} = \{Open, Close\}$
$OP_{FrontDoor} = \{Lock, Unlock\}$
$OP = OP_{TV} \cup OP_{PlayStation} \cup OP_{Oven} \cup OP_{Fridge} \cup OP_{FrontDoor}$
$OPA = \{KidsFriendly\}$
$KidsFriendly : op : OP \rightarrow \{True, False\}$
$KidsFriendly(G) = KidsFriendly(A3) = KidsFriendly(A7) = True$
$KidsFriendly(PG) = KidsFriendly(A12) = KidsFriendly(BuyGames) = False$
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 4.2: HABAC Use Case Configuration

(Fridge) unconditionally. (d) Allow teenagers to use entertainment devices unconditionally. (e) Allow parents to use any operation in any device unconditionally.

To achieve these objectives, HABAC can be configured as shown in Figure 4.2. Here, we have three users, $alex, bob,$ and $anne$ with user attribute $Relationship$ set to the values kid, $parent$, and $teenager$ respectively. When a user creates a session s, this session will automatically inherit subset of $user(s)$ attributes. We have 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 $Oven \leftarrow (DangerouseKitchenDevices : True)$. We have twelve operations G, PG, $A3$, $A7$, $A12$, $BuyGames$, ON, OFF, $Open$, $Close$, $Lock$, and $Unlock$. These operations are assigned to operation attributes as following, (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. We have one environment state $current$ which has three attribute functions (day, $time$, and $ParentInKitchen$).

The authorization function is a disjunction of five propositional statements. The first statement gives kids access to kids friendly operations during weekdays evenings, or weekends afternoon and evenings. The second statement gives teenagers access to dangerous kitchen devices only when one of the parents is in the kitchen. The third statement authorizes teenagers to use non dangerous kitchen devices unconditionally. The fourth statement allows teenagers to access kids friendly operations, and non kids friendly operations unconditionally. Finally, the fifth statement gives parents access to anything unconditionally.

4.3 Constructing HABAC From EGRBAC

In this section, we take a further step toward comparing HABAC, and EGRBAC. We introduce HABAC configuration that translates EGRBAC policies in a manner that they can be implemented by HABAC. The purpose is to see whether we can fully express any EGRBAC configuration in

Table 4.3: EGRBAC Configuration in HABAC

- $U_{HABAC} = U_{EGRBAC}$
- $USA_{HABAC} = \{Relationship\}$
- $Range(Relationship) = R_{EGRBAC}$
- $Relationship : u \in U_{HABAC} \rightarrow 2^{R_{EGRBAC}}$
- $Relationship : s \in S_{HABAC} \rightarrow 2^{R_{EGRBAC}}$
- $(\forall u_i \in U_{HABAC})[Relationship(u_i) = \{r_x | (u_i, r_x) \in UA_{EGRBAC}\}]$

- $UAConstraint_{HABAC} = \{uac_i\}$, where:
- $\forall(ssdc_i = (r_i, R_j) \in SSDConstraints_{EGRBAC})[uac_i = ((Relationship, r_i), UAP_j)]$, where:
  $UAP_j = \{(Relationship, r_n) | r_n \in R_j\}$

- $SAConstraint_{HABAC} = \{sac_i\}$, where:
- $\forall(dsdc_i = (r_i, R_j) \in DSDConstraints_{EGRBAC})[sac_i = ((Relationship, r_i), UAP_j)]$, where:
  $UAP_j = \{(Relationship, r_n) | r_n \in R_j\}$

- $ES_{HABAC} = \{Current\}$
- $ESA_{HABAC} = ER_{EGRBAC}$
- $(\forall esa_i \in ESA_{HABAC})[esa_i : es \in ES_{HABAC} \rightarrow \{True, False\}]$

- $D_{HABAC} = D_{EGRBAC}, OP_{HABAC} = OP_{EGRBAC}$
- $DA_{HABAC} = OPA_{HABAC} = DR_{EGRBAC}$
- $(\forall da_i \in DA_{HABAC})\ [da : d \in D_{HABAC} \rightarrow \{True, False\}]$
- $(\forall opa_i \in OPA_{HABAC})[opa : op \in OP_{HABAC} \rightarrow \{True, False\}]$
- $(\forall(dr_y \in DR_{EGRBAC}, p_x \in \{p_i | (p_i, dr_y) \in PDRA_{EGRBAC}\}))[dr_y(p_x.op) = True, dr_y(p_x.d) = True]$

- Initialize the authorization function $Authorization(s : S_{HABAC}, op : OP_{HABAC}, d : D_{HABAC}, current : ES_{HABAC})$
- For each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA_{HABAC}$, we construct an authorization policy as following:

1. $SetOfESA = "TRUE".\ \forall(esa \in ER_i)[SetOfESA = SetOfESA + " \wedge " + "esa(current) = True"]$.
2. $CurrentAuth = "r_i" + " \in " + "Relationship(s)" + " \wedge " + "dr_i(op) = True" + " \wedge " + "dr_i(d) = True" + " \wedge " + "SetOfESA"$.
3. $Authorization(s : S_{HABAC}, op : OP_{HABAC}, d : D_{HABAC}, current : ES_{HABAC}) \leftarrow Authorization(s : S_{HABAC}, op : OP_{HABAC}, d : D_{HABAC}, current : ES_{HABAC}) + " \vee " + "CurrentAuth"$.

HABAC model, and if not which model is more expressive, and in what terms.

The configuration of HABAC for a given EGRBAC configuration is shown in Table 4.3. In this configuration, for differentiation purposes every EGRBAC component is followed with the suffix $_{EGRBAC}$, similarly, every HABAC component is followed with the suffix $_{HABAC}$.

The goal is to construct HABAC configuration from EGRBAC configuration in such a way that the authorizations are the same as 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 $U_{HABAC}, USA_{HABAC}, ES_{HABAC}, ESA_{HABAC}$, $OP_{HABAC}, OPA_{HABAC}, D_{HABAC}, DA_{HABAC}, UAConstraint$, $SAConstraint$, and the authorization policy function $Authorization_{HABAC}(s : S, op : OP, d : D, es : ES)$.

The set of users, devices, and operations are the same in both systems. Roles are expressed through the user/session attribute $Relationship$ in HABAC. $Relationship$ is a user/session attribute that takes a user or a session as an input and returns the set of roles assigned to that user or that session. Static separation of duty constraints $SSDConstraints$ are translated into user attributes constraints in HABAC. Dynamic separation of duty constraints $DSDConstraints$ are translated into session attributes constraints in HABAC. Environment roles are translated into atomic environment state attributes. How to trigger different environment states attributes in response to environment's changes is outside the scope of this model. In EGRBAC device roles are ways of categorizing permissions. Since we do not have permission attributes in HABAC, and since a permission is a mapping between a device and an operation, we translate device roles in EGRBAC into atomic operation attributes and atomic device attributes with a range of values equal to $\{True, False\}$.

The final step is to translate the authorization policies. In EGRBAC it is the $RPDRA$ that gives specific role pairs and hence users access to specific device roles and hence permissions. Therefore, we translate each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA$ into an authorization policy. The final authorization policy is the disjunction of every created authorization policy.

As a result, we have only one user attribute which is $Relationship$. The number of user attributes constraints is equal to the number of $SSDConstraints$. The number of subject attributes constraints is equal to the number of $DSDConstraints$. The number of operation attributes, and the number of device attributes are equal to the number of device roles. The number of environment state attributes is equal to the number of environment roles.

In HABAC we can not create something equivalent to EGRBAC $PRConstraints$. As shown in Section 3.2.3, in EGRBAC, the way a user get access to specific set of permissions happens through a series of assignments, the most critical assignment is the $RPDRA$ which assigns role pairs to device roles. Hence, by controlling $RPDRA$ we can control which role pairs and hence roles get access to which device roles and hence permissions. In HABAC on the other hand, we don't have similar "attack point" that can be controlled to prevent specific access permission. The only way to make sure that certain users can not get access to specific permissions is by checking each request to access these permissions and look for those prohibited users. This is a significant advantage of EGRBAC in which we can enforce such constraints at assignment time whereas HABAC-like models would need to enforce these at enforcement time. To summarize, the construction of HABAC shown in Table 4.3 is equivalent to the given EGRBAC configuration including static and dynamic separation of duty constraints. The claim of equivalence is intuitively obvious since the construction is effectively one for one and straightforward. A formal argument can be presented along the lines of [115] but is tedious and does not provide meaningful insight.

4.4 Constructing EGRBAC from HABAC

In this section, we introduce our methodology to construct EGRBAC components and configurations from HABAC policy configuration. Our EGRBAC constructing approach works perfectly for HABAC policies that contain environment attributes, and static user/session, operation, and device attribute functions. It can also handles policies that do not involve comparing two different types of attributes. Furthermore, we found that due to some limitations in EGRBAC, it is either not possible to capture policies involving dynamic user/session, operation, and device attribute functions,

or costly (leads to role explosion).

We followed a bottom-up role engineering approach. Traditional algorithms for translating ABAC systems into RBAC using bottom-up approach, first represents the ABAC system in UPA (user-permission assignment) matrix, and then do some sort of role mining. The rows and columns of the matrix correspond to users and permissions, respectively. If a user is assigned a particular permission, the corresponding entry of the matrix contains a 1; otherwise, it contains a 0. From this perspective, role engineering is a process of matrix decomposition, wherein the Boolean matrix UPA is decomposed into two Boolean matrices (UA and PA), which together give the original access control policy [79].

EGRBAC is a more sophisticated model than RBAC; since in addition to user roles, it captures environment, and device characteristics through environment roles, and device roles respectively. In EGRBAC we do not give access to permissions directly, instead we give access to device roles. Hence, instead of UPA matrix, we first need to construct a user-device role assignment matrix, we call it user device role assignment array $UDRAA$. Moreover, users do not get access to authorized device roles unconditionally, some times specific set of environment roles need to be active. $UDRAA$ need to capture these information; each cell (u_i, dr_j) in $UDRAA$ contains zero if u_i cannot access dr_j, contains one if u_i can access dr_j unconditionally, or contains C if u_i can access dr_j when the set of conditions C is satisfied, were C is a set of user/session conditions (for example role), and environment conditions.

4.4.1 From Authorization policy to Authorization Array

An authorization policy is expressed in a logical clause. First we convert it into a disjunctive normal form (DNF). All logical formulas can be converted into an equivalent DNF form. Figure 4.3 shows the authorization policy of our HABAC use case which introduced in Figure 4.2 after following the standard approach to convert it into a DNF format.

We call each conjuncted term a condition. We have session, environment, device, operation, and mix conditions which are conditions that involve user/session, environment, device, operation,

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

Figure 4.3: The Authorization Policy of Use Case 1 in DNF Format

and any two types of attributes respectively.

In the DNF authorization policy of our HABAC use case mentioned in Figure 4.3 we have seven conjunctive clauses, each conjunctive clause represents one access authorization rule. To construct the authorization array we evaluate 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 which involve environment state attributes, we create a raw $(u_i, d_j, op_k, current, C)$ for that combination in the authorization array. Where, C is the set of session and environment related conditions in the examined conjunctive clause.

Authorizations array (AA): an authorization raw $(u_i, d_j, op_k, es_l, C)$ 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. The AA of the use case in Figure 4.2 is shown in Table 4.4. For illustration purposes each different color represents the authorization fields for different user.

4.4.2 Approach

The goal is to construct EGRBAC elements, assignments, and derived relations from HABAC policies in such a way that the authorizations are the same as those under HABAC. In this construction, for differentiation purposes every EGRBAC component is followed with the suffix $_{EGRBAC}$, similarly, every HABAC component is followed with the suffix $_{HABAC}$.

The inputs are HABAC set of users U_{HABAC}, set of devices D_{HABAC}, set of operations OP_{HABAC}, USA_{HABAC}, ESA_{HABAC}, OPA_{HABAC}, DA_{HABAC}, and the authorization array AA. The out-

Table 4.4: AA for HABAC Use Case

User u	Device d	Operation op	Environment state es	Conditions C
alex	TV	G	current	X
alex	PS	$A3$	current	X
alex	PS	$A7$	current	X
alex	TV	G	current	Z
alex	PS	$A3$	current	Z
alex	PS	$A7$	current	Z
bob	TV	G	current	$\{Relationship(s) = parent\}$
bob	TV	PG	current	$\{Relationship(s) = parent\}$
bob	PS	$A3$	current	$\{Relationship(s) = parent\}$
bob	PS	$A7$	current	$\{Relationship(s) = parent\}$
bob	PS	$A12$	current	$\{Relationship(s) = parent\}$
bob	PS	$BuyGames$	current	$\{Relationship(s) = parent\}$
bob	Oven	ON	current	$\{Relationship(s) = parent\}$
bob	Oven	OFF	current	$\{Relationship(s) = parent\}$
bob	Fridge	$Open$	current	$\{Relationship(s) = parent\}$
bob	Fridge	$Close$	current	$\{Relationship(s) = parent\}$
bob	FrontDoor	$Lock$	current	$\{Relationship(s) = parent\}$
bob	FrontDoor	$Unlock$	current	$\{Relationship(s) = parent\}$
anne	TV	G	current	$\{Relationship(s) = teenager\}$
anne	TV	PG	current	$\{Relationship(s) = teenager\}$
anne	PS	$A3$	current	$\{Relationship(s) = teenager\}$
anne	PS	$A7$	current	$\{Relationship(s) = teenager\}$
anne	PS	$A12$	current	$\{Relationship(s) = teenager\}$
anne	PS	$BuyGames$	current	$\{Relationship(s) = teenager\}$
anne	Oven	ON	current	Y
anne	Oven	OFF	current	Y
anne	Fridge	$OPEN$	current	$\{Relationship(s) = teenager\}$
anne	Fridge	$CLOSE$	current	$\{Relationship(s) = teenager\}$

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

Table 4.5: PDRA array for HABAC Use Case

	DangerouseKitchenDevices = True	DangerouseKitchenDevices = False	KidsFriendly = True	KidsFriendly = False	RemPerm
(TV, G)	0	0	1	0	0
(TV, PG)	0	0	0	1	0
$(PlayStation, A3)$	0	0	1	0	0
$(PlayStation, A7)$	0	0	1	0	0
$(PlayStation, A12)$	0	0	0	1	0
$(PlayStation, BuyGames)$	0	0	0	1	0
$(Oven, ON)$	1	0	0	0	0
$(Oven, OFF)$	1	0	0	0	0
$(Fridge, Open)$	0	1	0	0	0
$(Fridge, Close)$	0	1	0	0	0
$(FrontDoor, Lock)$	0	0	0	0	1
$(FrontDoor, Unlock)$	0	0	0	0	1

Table 4.6: UDRAA for HABAC Use Case

	DangerouseKitchenDevices = True	DangerouseKitchenDevices = False	KidsFriendly = True	KidsFriendly = False	RemPerm
alex	0	0	$\{X, Z\}$	0	0
bob	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$	$\{\{Relationship(s) = parent\}\}$
anne	$\{Y\}$	$\{\{Relationship(s) = teenager\}\}$	$\{\{Relationship(s) = teenager\}\}$	$\{\{Relationship(s) = teenager\}\}$	0

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

puts 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. The set of users, devices, and operations are the same in both systems, hence $U_{EGRBAC} = U_{HABAC}, D_{EGRBAC} = D_{HABAC}$, and $OP_{EGRBAC} = OP_{HABAC}$. 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 call it 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 device's or operation's attributes.

Step 3: Construct 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 4.5 for the $PDRA$ array of our HABAC use case.

Step 4: Construct 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 raw 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 raw 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. Note that these sets of conditions have to 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 4.6 shows $UDRAA$ for our use case.

Step 5: Construct the rest of EGRBAC elements $(R_{EGRBAC}, EC_{EGRBAC}, ER_{EGRBAC}, RP_{EGRBAC})$, assignments $(UA_{EGRBAC}, EA_{EGRBAC}, RPDRA_{EGRBAC})$, and derived relations $(RPRA_{EGRBAC}$, $RPEA_{EGRBAC})$ by following our proposed EGRBAC users and environment roles constructing algorithm introduced in Section 4.4.3. The set of users roles R_{EGRBAC} constructed here is the set of candidate users roles.

Step 6: Merge similar users roles. To do so, we run our developed role merging algorithm illustrated in Section 4.4.4.

4.4.3 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 following:

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 4.6 for our HABAC use case. Each column

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 raw i in $UDRAA$.
Require: $ContainsSC(X)$: Return True if the set of conditions X contains at least one session condition.
Require: $ContainsESC(X)$: Return True if the set of conditions X contains at least one environment state condition.
Require: $IsESC(c)$: Return True if c is an environment state condition.
Require: $ToString(x)$: Return the value of x in a string format.

Initilize n = Number of users, m = Number of device roles, $R_{EGRBAC} = \{\}$, $UA_{EGRBAC} = \{\}$, $EC_{EGRBAC} = \{\}$, $ER_{EGRBAC} = \{\}$, $EA_{EGRBAC} = \{\}$, $RP_{EGRBAC} = \{\}$, and $RPDRA_{EGRBAC} = \{\}$,
for $j \leftarrow 1$ to m **do**
  for $i \leftarrow 1$ to n **do**
    if $UDRAA[i,j] = 1$ **then**
      $er_x = Any\_Time$, $ec_x = True$
      $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_x\}$, $ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_x\}$
      $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_x\}, er_x)\}$
      $SER = \{er_x\}$
      $r_m = ToString(ColumnDR(j))$
      $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
      $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup \{(rp_z, ColumnDR(j))\}$
      $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
    else if $UDRAA[i,j] \neq 1 \wedge UDRAA[i,j] \neq 0$ **then**
      for each $X \in UDRAA[i,j]$ **do**
        if $(\neg ContainsSC(X) \wedge ContainsESC(X)$ **then**
          $SER = \{\}$
          for each $y \in X$ **do**
            Create ec_y, and er_y
            $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_y\}$, $ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_y\}$
            $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_y\}, er_y)\}$
            $SER = SER \cup \{er_y\}$
          end for
          $r_m = ToString(ColumnDR(j)) +" \wedge "+ ToString(X)$
          $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
          $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
          $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup (rp_z, ColumnDR(j))$
          $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
        else if $(ContainsSC(X) \wedge \neg ContainsESC(X)$ **then**
          $er_x = Any\_Time$, $ec_x = True$
          $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_x\}$, $ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_x\}$
          $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_x\}, er_x)\}$
          $SER = \{er_x\}$
          $r_m = ToString(ColumnDR(j)) +" \wedge "+ ToString(X)$
          $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
          $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
          $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup (rp_z, ColumnDR(j))$
          $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$

39: **else if** $(ContainsSC(X) \wedge ContainsESC(X))$ **then**
40: $SER = \{\}$
41: **for each** $y \in \{y|(y \in X) \wedge IsESC(y)\}$ **do**
42: Create ec_y, and er_y
43: $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_y\}$, $ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_y\}$
44: $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_y\}, er_y)\}$
45: $SER = SER \cup \{er_y\}$
46: **end for**
47: $r_m = ToString(ColumnDR(j)) + " \wedge " + ToString(X)$
48: $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
49: $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
50: $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup (rp_z, ColumnDR(j))$
51: $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
52: **end if**
53: **end for**
54: **end if**
55: **end for**
56: **end for**

is corresponding to users access rights to a specific device role. Inside each column loop through the fields of different rows. Here we have two cases:

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

a 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, 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 $SER = \{er_x\}$.

b Creates 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}.

c Defines a role pair rp_z, where $rp_z.r = r_m$ and $rp_z.ER = SER$. Add rp_z to the set RP_{EGRBAC}.

d Assigns the role pair rp_z to the device role corresponding to this column by adding the pair $(rp_z, ColumnDR(j))$ to the set $RPDRA_{EGRBAC}$.

e Assigns the role r_m to the user corresponding to this raw by adding the pair$(RawUser(i), r_m)$ to the set UA_{EGRBAC}.

B. $UDRAA[i,j] \neq 1 \wedge UDRAA[i,j] \neq 0$**,** which means that the user u_i can access the device role dr_j under specific set of user and environment conditions defined by $UDRAA[i,j]$. Here, $UDRAA[i,j]$ is a set of sets of conditions, each set of conditions define a group of session, and environment conditions that need to be satisfied together in order for the user u_i to be able to access the device role dr_j. Loop through each set of conditions $X \in UDRAA[i,j]$, X satisfies only one of the three following options:

1. X contains environment conditions only. In this case the algorithm first loops through each environment condition to create a corresponding environment condition ec_y, environment role er_y, and add these environment roles to the set SER. Second, the algorithm creates a corresponding role which represents accessing this column device role when the set of environment attributes conditions that form X is satisfied. Finally, it follows the same three steps c, d, and e explained in the previous case (when $UDRAA[i,j] = 1$).

2. X contains session conditions only. Here the algorithms follows the same steps explained in case A (when $UDRAA[i,j] = 1$). The only difference here is that the created role corresponds to access this column device role anytime, under the set of user conditions expressed by X instead of unconditionally as in case A.

3. X contains session and environment conditions. In this case the algorithm creates corresponding environmnet roles, environment conditions, and user role. It then follows the same three step c, d, and e explained in case A.

4.4.4 Users Roles Merging Algorithm

The main purpose of this algorithm is to merge roles that have similar users assignments. For each two roles r_i, r_j which are assigned to the same set of users, the algorithm does the following:

$r_1 \equiv DangerouseKitchenDevices = True \wedge Relationship(s) = parent,$
$r_2 \equiv DangerouseKitchenDevices = True \wedge \{Relationship(s) = teenager,$
$\quad ParentInKitchen(current) = True\},$
$r_3 \equiv DangerouseKitchenDevices = False \wedge Relationship(s) = parent\ ,$
$r_4 \equiv DangerouseKitchenDevices = False \wedge Relationship(s) = teenager\ ,$
$r_5 \equiv KidsFriendly = True \wedge \{Relationship(s) = kid,$
$\quad day(current) \in \{Sa, S\}, 12:00 \leq time(current) \leq 19:00\},$
$r_6 \equiv KidsFriendly = True \wedge \{Relationship(s) = kid,$
$\quad day(current) \in \{M, T, W, Th, F\}, 17:00 \leq time(current) \leq 19:00\},$
$r_7 \equiv KidsFriendly = True \wedge Relationship(s) = parent,$
$r_8 \equiv KidsFriendly = True \wedge Relationship(s) = teenager,$
$r_9 \equiv KidsFriendly = False \wedge Relationship(s) = parent,$
$r_{10} \equiv KidsFriendly = False \wedge Relationship(s) = teenager,$
$r_{11} \equiv RemPerm \wedge Relationship(s) = parent$

Figure 4.4: Initial Set of Roles Before Running the Role Merging Algorithm

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 $RPDRA_{EGRBAC}$. See Algorithm 2 for the complete algorithm.

After applying the first five steps of the approach of constructing EGRBAC from HABAC introduced in Section 4.4.2 on our HABAC use case, we will end up having a set of eleven roles as illustrated in Figure 4.4. These roles will be assigned to different users as following:

$UA = \{(alex, r_5), (alex, r_6), (bob, r_1), (bob, r_3), (bob, r_7), (bob, r_9), (bob, r_{11}),$
$\quad (anne, r_2), (anne, r_4), (anne, r_8), (anne, r_{10})\}.$

After running the users roles merging algorithm, the constructed eleven roles will be merged into three roles only, and the user role assignment set will end up having three pairs as shown in the following:

$R = \{\ r_a \equiv r_1, r_3, r_7, r_9, r_{11}, r_b \equiv r_2, r_4, r_8, r_{10}, r_c \equiv r_5, r_6\}.$

$UA = \{(alex, r_c), (bob, r_a), (anne, r_b)\}.$

Algorithm 2 Users Roles Merging 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. .

for each $r_i, r_j \in R_{EGRBAC}$ **do**
  if $U(r_i) = U(r_j)$ **then**
    for each $rp_k \in RP(r_i)$ **do**
      $rp_k.r = r_j$
    end for

    $R_{EGRBAC} = R_{EGRBAC} \setminus r_i$

    ▷ Delete all UA pairs related to r_i
    for each $(u_l, r_i) \in UA_{EGRBAC}$ **do**
      $UA = UA \setminus (u_l, r_i)$
    end for

    ▷ Replace all $RPDRA$ pairs related to r_i
    for each $((r_i, ER_x), dr_y) \in RPDRA_{EGRBAC}$ **do**
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \setminus ((r_i, ER_x), dr_y)$
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup \{((r_j, ER_x), dr_y)\}$
    end for
  end if
end for

4.4.5 The output of EGRBAC Constructing Approach on HABAC Use Case

The output of EGRBAC role constructing algorithm for the Use case in Table 4.2 is shown in Table 4.7. 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, maximum number of environment roles and conditions is $\Omega(|ESA|)$. Finally, maximum number of user roles is $O(2^{|SA|+|ESA|})$.

4.5 Analysis and Limitations

Our proposed HABAC model is a user to device access control model. It captures different users, environment, operations, and devices characteristics. Therefore, it is a dynamic model. It is a fine grained model; since it is capable of giving users access to some operations within a single device

without the need to give them access to the entire device.

Our approach of constructing HABAC from EGRBAC is a simple, straightforward approach that is capable of translating EGRBAC configuration into an HABAC policy configuration. However, as we discussed in Section 4.3, in HABAC we can not create something equivalent to EGRBAC $PRConstraints$. This makes it troublesome to prevent future authorization of specific users to access specific operations on specific devices, since the only way to do so is dynamically at enforcement time when the user is trying to access the prohibited operation, unlike EGRBAC in which we can enforce this prevention at assignment time. In EGRBAC, determining the role structure could take a lot of efforts, but when completed it is easy to define who has what permissions, and who is not allowed to have a future access to specific permissions. On the other hand, in HABAC this is not achievable.

In addition to users/sessions, devices, and operations static attributes, our EGRBAC constructing approach is capable of handling HABAC policies that contain environment attributes. Due to some limitations in EGRBAC, our approach can't handle HABAC policies that involve users/sessions or devices dynamic attributes. As explained in Section 4.2.1, dynamic attributes are those attributes that are rapidly changing without involving administration actions. For instance device temperature. In our approach, we translate device, and permission attributes instances into device roles. In EGRBAC, device roles are means of categorizing permissions of different devices according to relatively static characteristics. when a permission is assigned to a specific device role, then it is part of that device role until some administration change happens, there is no way to dynamically activates and deactivates neither device roles, nor assignment of permissions to different device roles. In HABAC 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 one is to create two device roles one for high temperature, and another for low temperature for each device. for a large number of devices and dynamic attributes this option may lead to a role explosion. Furthermore, there is no mechanism in EGRBAC

that can dynamically activates 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. The second option is for those devices which have similar access conditions we create a device role for low temperature, and a device role for high temperature. However, there is no way to dynamically activates or deactivates devices membership in different device roles according to their temperatures. The similar argument holds when we deal with dynamic user attributes.

Our EGRBAC constructing approach didn't consider the following: (1) Policies that compare two different types of attributes. (2) HABAC configurations that involve user attributes constraints and session attributes constraints.

From the above, a hybrid model combining HABAC and EGRBAC features may be the most suitable for smart home IoT, and likely more generally.

Table 4.7: The output of EGRBAC Constructing Approach on HABAC Use Case

(a) $U_{EGRBAC} = U_{H-ABAC}, D_{EGRBAC} = D_{H-ABAC}, OP_{EGRBAC} = OP_{H-ABAC}$, $P_{EGRBAC} =$
$\{(TV, G), (TV, PG), (PlayStation,$
$A3), (PlayStation, A7), (PlayStation, A12), (PlayStation,$
$BuyGames), (Oven, ON), (Oven, OFF), (Fridge, Open), (Fridge,$
$Close), (FrontDoor, Lock), (FrontDoor, Unlock)\}$
(b) $DR = \{DangerouseKitchenDevices = True,$
$DangerouseKitchenDevices = False,$
$KidsFriendly = True\ , KidsFriendly = False,$
$RemPerm\}.$
(c) $PDRA = \{((TV, G), KidsFriendly = True),$
$(PlayStation, A3), KidsFriendly = True),$
$((PlayStation, A7), KidsFriendly = True),$
$((TV, PG), KidsFriendly = False),$
$((PlayStation, A12), KidsFriendly = False),$
$((PlayStation, BuyGames), KidsFriendly = False),$
$((Oven, ON), DangerouseKitchenDevices = True),$
$((Oven, OFF), DangerouseKitchenDevices = True),$
$((Fridge, Open), DangerouseKitchenDevices = False),$
$((Fridge, Close), DangerouseKitchenDevices = False),$
$((FrontDoor, Lock), RemPerm),$
$((FrontDoor, Unlock), RemPerm)\}.$
(d) $EC = \{True,$
$ec_1 \equiv ParentInKitchen(current) = True,$
$ec_2 \equiv day(current) \in \{Sa, S\},$
$ec_3 \equiv 12:00 \leq time(current) \leq 19:00\ ,$
$ec_4 \equiv day(current) \in \{M, T, W, Th, F\}\ ,$
$ec_5 \equiv 17:00 \leq time(current) \leq 19:00\}.$
(e) $ER = \{Any\_Time,$
$er_1 \equiv ParentInKitchen(current) = True,$
$er_2 \equiv day(current) \in \{Sa, S\} \equiv Weekend,$
$er_3 \equiv 12:00 \leq time(current) \leq 19:00 \equiv Afternoon\ and\ Evening,$
$er_4 \equiv day(current) \in \{M, T, W, Th, F\} \equiv Weekdays,$
$er_5 \equiv 17:00 \leq time(current) \leq 19:00 \equiv Evening\}.$
(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) $UA = \{(bob, r_a), (anne, r_b), (alex, r_c)\}.$
(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) $RPDRA = \{((r_a, Any_Time)\ , DangerouseKitchenDevices = True),$
$((r_a, Any\_Time)\ , DangerouseKitchenDevices = False),$
$((r_a, Any\_Time)\ , KidsFriendly = True),$
$((r_a, Any\_Time)\ , KidsFriendly = False),$
$((r_a, Any\_Time)\ , RemPerm),$
$((r_b, \{er_1\})\ , DangerouseKitchenDevices = True),$
$((r_b, Any\_Time)\ , DangerouseKitchenDevices = False),$
$((r_b, Any\_Time)\ , KidsFriendly = True),$
$((r_b, Any\_Time)\ , KidsFriendly = False),$
$((r_c, \{er_2, er_3\})\ , KidsFriendly = True),$
$((r_c, \{er_4, er_5\})\ , KidsFriendly = True)\}.$

CHAPTER 5: HYBRID ATTRIBUTE AND ROLE BASED ACCESS CONTROL MODELS FOR SMART HOME IOT (HYBAC$_{RC}$ AND HYBAC$_{AC}$)

This chapter presents two hybrid models for smart home IoT access control. Here, we utilize two distinct approaches to develop two different hybrid models. We followed a role-centric approach and an attribute-centric approach to develop $HyBAC_{RC}$ and $HyBAC_{AC}$ respectively. We formally define these models and illustrate their features through use case scenarios. We further provide a proof-of-concept implementation for each model in Amazon Web Services (AWS) IoT platform. Moreover, we compare the theoretical expressive power of $HyBAC_{AC}$, and $HyBAC_{RC}$ models by providing algorithms for converting an $HyBAC_{AC}$ specification to $HyBAC_{RC}$ and vice versa. Finally, we provide a comprehensive theoretical comparison between the four smart home IoT access control models introduced in this dissertation.

5.1 Motivation

In role based access control (RBAC) models [48, 100] permissions are associated with roles, and users are made members of appropriate roles, thereby acquiring the roles' permissions. On the other hand, in attribute based access control (ABAC) models [63, 66], access is granted according to attributes associated with the user and resource. While ABAC may require up to 2^n rules for n attributes, attempting to implement the same controls in RBAC could, in a worst case, require 2^n roles, one for each possible combination of attributes. Hence, RBAC trades up-front role structuring effort (role engineering) for ease of administration and user permission review, while ABAC makes the reverse trade-off: it is easier to set up, but analyzing or changing user permissions can be problematic [70].

A dynamic, expressive, and accurate enough access control model has to capture different users, devices, operations, and environment characteristics (attributes) beyond user roles. In general, we have two types of attributes that need to be expressed and used in authorization policies:

static attributes, and dynamic attributes [70]. Static Attributes are relatively static and change only by administrative action. For example, user relation to the house, skill set and danger level of device operations. Dynamic Attributes change due to different external conditions, possibly rapidly and unpredictably. For instance, time, user location, weather, and device temperature. Expressing dynamic attributes in RBAC models can be costly and cumbersome.

In Chapter 3, we presented EGRBAC, which is an extended version of RBAC to cover different users, devices, permissions, and environment attributes. EGRBAC and similar RBAC based models can cover static attributes, and dynamic environment attributes effectively. On the other hand, dynamic attributes specific to individual users and devices (objects) create significant difficulties for RBAC, for two principal reasons. The multiplicity of combinations that need to be considered can lead to role explosion. Moreover, RBAC lacks mechanisms to dynamically activate and deactivate different user, session and roles according to varying dynamic characteristics. For example, consider that the home owner wants to grant teenagers use of dangerous permission for kitchen devices, such as opening the oven, but only when the oven temperature is below a threshold, say 250°. This use case can easily be handled in ABAC models such as in HABAC [20] introduced in Chapter 4, by defining a dynamic device attribute $device\_temperature$ which measures the device temperature, and then configuring an access policy that authorizes teenagers to such dangerous permissions only if this attribute is below 250°. We could try and do this in EGRBAC by defining two roles for each such device, one for high temperature and the other for low temperature. To reduce the role explosion we could aggregate high temperature sensitive devices into a single high temperature and single low temperature role for all such devices. Nevertheless, we would still need to add mechanism to EGRBAC to dynamically activate or deactivate devices membership in different device roles according to their temperatures [20]. A similar situation arises when we deal with dynamic user attributes. For example, suppose we want to permit teenagers to use the front door lock permissions in some special circumstances when they are granted a token by one of the parents. In RBAC systems we could construct different roles for different token values for each teenager. To reduce the role explosion we could instead define a single role for each token

value for all teenagers. As above we would also need to add mechanism to the RBAC system to dynamically activate or deactivate users membership in different token value roles.

On the other hand, in ABAC based models including HABAC [20] it is easy to specify access rules, but to determine the permissions available to a particular user, a potentially large set of rules might need to be executed in exactly the same order in which the system applies them. This can make it practically impossible to determine risk exposure for a given employee position [70]. Moreover, it is troublesome to prevent future authorization of specific users to access specific operations on specific devices, since the only way to do so is dynamically at enforcement time when the user is trying to access the prohibited operation, unlike EGRBAC in which we can enforce such prevention at assignment time.

From the above, a hybrid model combining HABAC and EGRBAC features may be the most suitable for smart home IoT, and likely more generally.

5.1.1 Combining RBAC and ABAC

Kuhn et al [70] specified three approaches for combining ABAC, and RBAC concepts to use both roles and attributes in authorization decisions. These approaches are: dynamic roles, attribute-centric, and role-centric. In the dynamic roles approach, attributes such as time of day are used by a front-end module to determine the user's role, retaining a conventional role structure but changing user-role assignment dynamically. In the attribute-centric approach, a role name is just one of many attributes with no special semantics. In contrast with conventional RBAC, the role is not a collection of permissions but the name of an attribute called "role." Finally, in the role-centric approach, attributes are added to constrain RBAC. Constraint rules that incorporate attributes can only reduce permissions available to the user via roles, but not expand them. In this research we used the role-centric, and attribute-centric approaches to develop two different hybrid models.

The reason for avoiding dynamic roles approach is that home IoT environment is rich with attributes. Having so many combinations of attributes values may result in large numbers of user roles, and device roles, which will further complicate the model, and the implementation. Such

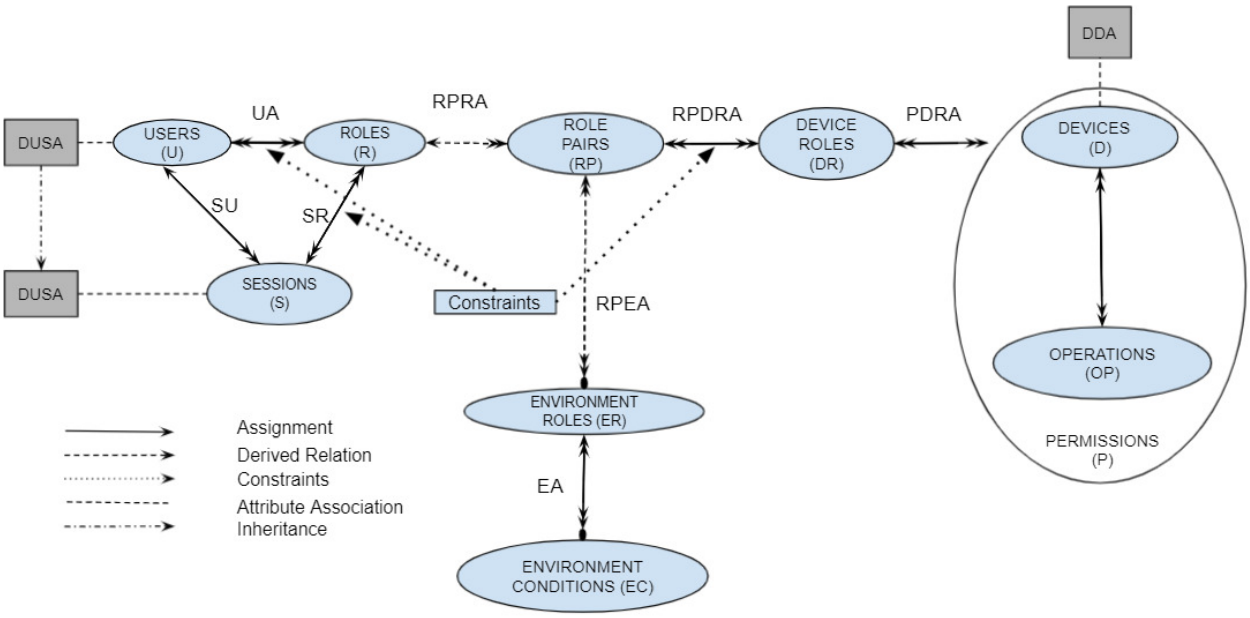

Figure 5.1: Smart Home IoT $HyBAC_{RC}$ Model

approach may better fit environment with few dynamic attributes, for instance, in the case of smart offices where in addition to the static attributes which effect permission authorization decisions, such as user's position, we may also have few dynamic attributes which contribute in authorization decisions. For instance, time, and access location.

5.2 $HyBAC_{RC}$ model

In this section, we introduce the $HyBAC_{RC}$ model. The model is conceptually depicted in Figure 5.1 while formal definitions are given in Tables 5.1 and 5.2. $HyBAC_{RC}$ fully incorporates the EGRBAC model [19] (which was introduced in Section 3.2) while adding dynamic attributes for users, sessions, and devices. The components of EGRBAC are shown in blue while the added dynamic attributes are shown in grey boxes in Figure 5.1.

$HyBAC_{RC}$ adopts a role-centric design [70]. In this approach, relatively static attributes (for users, sessions, and devices) define the user's and device's roles. On the other hand, dynamic attributes define attribute-based rules within the scope of the relatively static roles. In this way, we avoid both problems of role explosion and dynamic role activation and deactivation. Developing a

role structure based on the more static attributes can avoid awkward designs that might result from purely one choice or another [70]. For example, consider a system with 10 users attributes, four of which are static and six are dynamic. In the worst case, this could result in 2^{10} roles in RBAC based models (including EGRBAC) or 2^{10} rules in ABAC based models (including HABAC). Establishing a policy structure based on the four static and six dynamic attributes amounts to a worst case of 16 roles and 64 rules, while also effectively separating the policy concerns of the relatively static roles and the intrinsically dynamic attributes. The potential explosion of roles or rules will clearly get worse with device attributes also included, reinforcing our argument for a hybrid approach.

5.2.1 $HyBAC_{RC}$ Formal definition

The basic components of $HyBAC_{RC}$ are discussed below.

EGRBAC basic sets:

All EGRBAC components are incorporated into $HyBAC_{RC}$, as shown in blue in Figure 5.1 with the definitions given in the top portion of Table 5.1. Roles (R) are similar to the traditional RBAC user's roles. However, in smart homes, a role specifically represents the relationship between the user and the family, which encompasses parents, kids, babysitters, and such [59]. Device roles (DR) are means of categorizing permissions of different devices. For example, we can categorize the dangerous permissions of various smart devices by creating a device role called dangerous permissions and assigning dangerous permissions (such as turning on the oven, turning on the mower) to it. Environment roles (ER) represent environmental contexts, such as daytime/nighttime. Environment roles are turned on/off (i.e., triggered) by subsets of Environment Conditions (EC) such as daylight, these environment conditions need to be active together to trigger on specific environment role assigned to it through the EA relation. A role pair rp has a role part $rp.r$ that is the single role associated with rp, and an environment role part $rp.ER$ is the subset of environment roles associated with rp. The main idea in EGRBAC is that a user is assigned a subset of roles and, according to the current active roles in a session and current active environment conditions

(which determine the current active environment roles), some role pairs will be active, whereby the user will get access to the permissions assigned to the device roles which are assigned to the current active role pairs. However, in $HyBAC_{RC}$ this structure of assignments only define the maximum permissions available to a specific user in general, whereas defining which permissions among these maximum permissions are currently available for this user is further determined by the current values of the dynamic user and session attributes and dynamic device attributes.

EGRBAC Derived Functions:

These are three derived functions that are useful to define for EGRBAC. First, the function $roles(s)$ which takes a session s as an input and returns the set of roles assigned to s. Second, the function $users(s)$ takes a session s as an input and returns the unique user who created s (constant for the session lifetime). Third, the function $droles(p)$ takes permission p as an input and returns the set of device roles assigned to p.

EGRBAC Constraints:

These are the same three EGRBAC constraint as follows.

i Permission-role constraints (PRConstraints) prevent assignments that would enable specific roles to access specifically prohibited permissions.

ii Static Separation of Duty (SSD) is the familiar SSD in RBAC. It enforces constraints on the assignment of users to roles. In other words, if a user is authorized as a member of one role, the user is prohibited from being a member of a second conflicting role [99].

iii 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 in independently, but produce policy concerns when allowed to be acted simultaneously in the same session [99].

Dynamic attributes:

These additions to EGRBAC are shown as grey rectangles in Figure 5.1 with the definitions given in the middle portion of Table 5.1. Attributes are characteristics that are used in access control

decisions. Formally, an attribute is a function that takes an entity such as a user and returns a specific value from its range. An attribute range is given by a finite set of atomic values. An atomic valued attribute will return one value from the range, while a set valued attribute will return a subset of the range. As discussed in Section 5.1, dynamic attributes change due to different conditions rather than as a result of administrative actions. An example for users dynamic attributes can be user location, user temporal health condition, user awaking status, etc. Similarly, for devices, we may be interested in device location, device temperature, device usage status, etc.

Dynamic users and sessions attributes ($DUSA$) is the set of dynamic attributes associated with both users and sessions. Each session s inherits a subset of the dynamic attributes of its unique user creator, this is controlled by the unique user creator $user(s)$. If a session s inherited a dynamic user session attribute $dusa$ from his user $user(s)$, then it is required that $dusa(s) = dusa(user(s))$.

Dynamic device attributes (DDA) is the set of dynamic attributes associated with devices. In $HyBAC_{RC}$ we do not consider environment and operations dynamic attributes for the following reasons. First, regarding the element environment roles (ER), the way it is designed and associated with the role pairs (RP) element enables it to captures both static and dynamic environment attributes; a role pair will be active only if the set of associated environment roles are currently active and triggered by their corresponding environment conditions. Second, regarding operations (OP), they are basically defined by the manufacturer and usually have a static nature such as dangerous or benign.

Attributes authorization function (rules):

This is a logic formula which is evaluated for each access decision. It is defined using the grammar of Table 5.2 which is evaluated for each access decision. For a specific session s_i, device d_j and operation op_k the authorization function $Authorization(s_i, op_k, d_j)$ is evaluated by substituting the actual attribute values of $dusa(s_i)$, $dda(d_j)$, $roles(s_i)$ and $droles((op_k, d_j))$ for the corresponding symbolic placeholders and evaluating the resulting logical formula to be True or False.. In Table 5.2 we define a policy language for attributes authorization functions using propositional logic formula.

Table 5.1: $HyBAC_{RC}$ Model Formalization Part I: Basic Sets and Dynamic Attributes

Users, Roles and Sessions

$-U$ and R are sets of users and roles respectively (home owner specified)

$-UA \subseteq U \times R$, many to many user role assignment relation (home owner specified)

We define the derived function $roles(u) : U \rightarrow 2^R$, where: $roles(u_i) = \{r_j \mid (u_i, r_j) \in UA\}$

$-S$ is the set of sessions (each session is created, terminated and controlled by an individual user)

$-SU \subseteq S \times U$, many to one relation assigning each session to its single controlling user

We define the derived function $user(s) : S \rightarrow U$, where: $user(s_i) = u_j$ such that $(s_i, u_j) \in SU$

$-SR \subseteq S \times R$, many to many relation that assigns each session to a set of roles that can be changed by the controlling user

We define the derived function $roles(s) : S \rightarrow 2^R$, where: $roles(s_i) = \{r_j \mid (s_i, r_j) \in SR\}$

It is required that $roles(s) \subseteq roles(user(s))$ at all times

Devices, Operations, Permissions and Device Roles

$-D$ is the set of devices deployed in the smart home (home owner deployed)

$-OP$ and $P \subseteq D \times OP$ are sets of operations and permissions respectively (device manufacturers specified)

$-DR$ is the set of device roles (home owner specified)

$-PDRA \subseteq P \times DR$, many to many permissions to device roles assignment (home owner specified)

We define the derived function $droles(p) : P \rightarrow 2^{DR}$, where: $droles(p_i) = \{dr_j \mid (p_i, dr_j) \in PDRA\}$

Environment Conditions and Environment Roles

$-EC$ is the set of boolean environment conditions (determined by sensors deployed in the smart home under home owner control)

At any moment each $ec_i \in EC$ is either True or False depending on the state of the corresponding sensor

$-ER$ is the set of environment roles (home owner specified)

$-EA \subseteq 2^{EC} \times ER$, many to many environment role activation relation (home owner specified)

At any moment, $er \in ER$ is activated iff $(\exists(ec_{i1}, ec_{i2}, \ldots, ec_{in}), er) \in EA)[ec_{i1} \wedge ec_{i2} \wedge \ldots \wedge ec_{in} = \text{True}]$ at that moment

Role Pairs

$-RP \subseteq R \times 2^{ER}$, many to many role pairings of user role and subsets of environment roles (home owner specified)

For $rp = (r_i, ER_j) \in RP$, we define $rp.r = r_i$ and $rp.ER = ER_j$

We define the derived relation $RPRA \subseteq RP \times R$ where: $RPRA = \{(rp_m, r_n) \mid rp_m \in RP \wedge rp_m.r = r_n\}$

We define the derived relation $RPEA \subseteq RP \times 2^{ER}$ where: $RPEA = \{(rp_m, ER_n) \mid rp_m \in RP \wedge ER_n = rp_m.ER\}$

Role Pair Assignment

$-RPDRA \subseteq RP \times DR$, many to many RP to DR assignment (home owner specified)

Constraints

$-PRConstraints \subseteq 2^P \times 2^R$, many to many permission-role constraints relation (home owner specified)

For each $(P_i, R_j) \in PRConstraints$ it is required that

$(\forall p_m \in P_i)(\forall r_n \in R_j)(\forall(rp_p, dr_q) \in RPDRA)[(p_m, dr_q) \notin PDRA \vee rp_p.r \neq r_n]$

$-SSDConstraints \subseteq R \times 2^R$, many to many static separation of duty constraints relation (home owner specified)

For each $(r_i, R_j) \in SSDConstraints$ it is required that $(\forall u \in U)(\forall r \in R_j)[(u, r) \in UA \implies (u, r_i) \notin UA]$

$-DSDConstraints \subseteq R \times 2^R$, many to many dynamic separation of duty constraints relation (home owner specified)

For each $(r_i, R_j) \in DSDConstraints$ it is required that $(\forall s \in S)(\forall r \in R_j)[(s, r) \in SR \implies (s, r_i) \notin SR]$

Dynamic Attributes

$-DUSA$ and DDA are finite sets of dynamic user and session attribute functions and dynamic device attribute functions respectively, where $DUSA \cap DDA = \emptyset$ for convenience (determined by sensors deployed in the smart home under home owner control)

$-$Each session s inherits a subset of the dynamic attribute functions of its unique user creator (controlled by the session creator $user(s)$)

For every inherited attribute function $att \in DUSA$, $att(s) = att(user(s))$ at all time

For every non-inherited attribute function $att \in DUSA$, $att(s)$ is undefined and its use in any logical formula renders that formula false

$-$For each $att \in DUSA \cup DDA$, $Range(att)$ is the attribute range which is a finite set of atomic values

$-attType : DUSA \cup DDA \rightarrow \{set, atomic\}$.

$-$Each $att \in DUSA \cup DDA$ correspondingly maps users in U or sessions in S, or devices in D to atomic or set attribute values. Formally:

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

At any moment, the value of att for a given user or device is automatically determined by sensors deployed in the smart home

Attributes Authorization Function

$-Authorization(s : S, op : OP, d : D)$ is a logic formula defined using the grammar of Table 5.2 (home owner specified)

It is evaluated for a specific session s_i, device d_j and operation op_k as specified in Table 5.2

CheckAccess Predicate

$-CheckAccess$ is evaluated when session s_i attempts operation op_k on device d_j while the environment conditions in EC_l are True

$-CheckAccess(s_i, op_k, d_j, E_l)$ evaluates to True or False using the following formula:

$$Authorization(s_i, op_k, d_j) \wedge$$
$$(\exists(rp_m, dr_n) \in RPDRA)\ [((d_j, op_k), dr_n) \in PDRA \wedge (s_i, rp_m.r) \in SR \wedge$$
$$rp_m.ER \subseteq \{er \in ER \mid (\exists EC'_l \subseteq EC_l)[(EC'_l, er) \in EA]\}]$$

Table 5.2: $HyBAC_{RC}$ Model Formalization Part II: Attributes Authorization Function

Attributes Authorization Function

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

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists x \in set.\alpha \mid \forall x \in set.\alpha \mid$
  $set\ setcompare\ set \mid atomic \in set \mid atomic \notin set \mid atomic\ atomiccompare\ atomic$
- $setcompare ::= \subset \mid \subseteq \mid \not\subseteq$
- $atomiccompare ::= < \mid = \mid \leq$
- $set ::= dusa(s) \mid dda(d) \mid roles(s) \mid droles((op, d))$, for $attType(dusa) = set$ and $attType(dda) = set$
- $atomic ::= dusa(s) \mid dda(d) \mid value$, for $attType(dusa) = atomic$ and $attType(dda) = atomic$

$-$For a specific session s_i, device d_j and operation op_k the authorization function $Authorization(s_i, op_k, d_j)$ is evaluated by substituting the actual attribute values of $dusa(s_i)$, $dda(d_j)$, $roles(s_i)$ and $droles((op_k, d_j))$ for the corresponding symbolic placeholders and evaluating the resulting logical formula to be True or False.

Check access predicate:

The bottom part of Table 5.1 formalizes the check access predicate of $HyBAC_{RC}$. Consider a session s_i which attempts to perform operation op_k on device d_j when the subset of environment conditions EC_l are active. This operation will succeed if and only if both of the following are true. First, the authorization function $Authorization(s_i, op_k, d_j)$ evaluates to TRUE. Second, the requirements of role membership and role activation specified by EGRBAC are true. These are specified in the bottom two lines of the authorization predicate in Table 5.1. The EGRBAC requirement can be stated in words as follows. There is a role pair rp_m and a device role dr_n assigned to each other in $RPDRA$ such that the following conditions are true. (i) dr_n is assigned to the permission (d_j, op_k) in $PDRA$. (ii) $rp_m.r$ is one of the active roles of s_i (as given in SR). (iii) Each environment role $er \in rp_m.ER$ is active because it is activated by a subset of the currently active environment conditions EC_l.

5.2.2 Use Case Demonstration

Here we present one use case scenario to demonstrate how to configure $HyBAC_{RC}$ components to enforce specific access control policies. This use case has the following objectives. (a) Authorize teenagers to use dangerous permissions in kitchen devices (open the oven, and turn on the oven)

only when a parent is in the kitchen and the current temperature of the oven is less than or equal to $250°$. (b) Authorize teenagers to use non dangerous permissions in kitchen devices (close the oven, turn off the oven, open and close the fridge) unconditionally. (c) Authorize teenagers to use the front door lock permissions (lock, and unlock) if they are currently temporarily granted those permissions by one of the parents. (d) Allow teenagers to use entertainment devices permissions during weekends evenings and nights if the accessed device is not in use by someone else. (e) Allow kids to use kids friendly operations in entertainment devices which are, turning on and off the device, and G content during weekends evenings only, and if the accessed device is not in use by someone else. (f) Finally, allow parents to use any operation in any device unconditionally.

For this use case $HyBAC_{RC}$ will be configured as shown in Figure 5.2, and Figure 5.3. First we configure the maximum permissions available for each user using EGRBAC components as follows. We have five users *bob*, $alex$, $suzanne$, $john$, and $anne$ respectively assigned to roles $parents$, $kids$, $kids$, $teenagers$, and $teenagers$. The devices include $Oven$, $Fridge$, $FrontDoorLock$, $PlayStation$, and TV. We have five device roles: $Dangerous\_Kitchen\_Permissions$, $Non\_Dangerous\_Kitchen\_Permissions$, $Front\_Door\_Lock$, $Entertainment\_Devices$, and $Kids\_Friendly\_Content$. We assign the oven permissions $\{On_{Oven}, Open_{Oven}\}$ to the device role $Dangerous\_Kitchen\_Permissions$. We assign the oven permissions $\{Off_{Oven}, Close_{Oven}\}$ and all fridge permissions to the $Non\_Dangerous\_Kitchen\_Permissions$ device role. Moreover, we assign all front door lock permissions to the device role $Front\_Door\_Lock$. All the permissions of the TV and the $PlayStation$ devices are assigned to $Entertainment\_Devices$ device role, and an appropriate subset of these permissions are assigned to $Kids\_Friendly\_Content$ device role. EC comprises $Parent\_Is\_In\_The\_Kitchen$, $weekends$, $evenings$, $nights$, and $TRUE$, respectively active when a parent is in the kitchen, on weekends, during evening, during night, and always. The environment role $Teenagers\_Kitchen\_Time$ is active when the environment condition $Parent\_Is\_In\_The\_Kitchen$ is active. The environment role $Kids\_Entertainment\_Time$ is active when both environment conditions $weekends$ and $evenings$ are active. Similarly, The environment role $Teenagers\_$

$Entertainment\_\ Time$ is active when $weekends$ and $evenings$ or $weekends$ and $nights$ environment conditions are active. The environment condition Any_Time is always active.

$RPDRA$ assignments specify that $kids$ can access the permissions assigned to the device role $Kids\_Friendly\_Content$ when $Kids\_Entertainment\_Time$ is active. $RPDRA$ assignments also specify that $teenagers$ can access the permissions assigned to $Dangerous\_Kitchen\_\ Permissions$ when $Teenagers\_\ Kitchen\_\ Time$ environment role is active, whereas they can access $Non\_\ Dangerous\_Kitchen\_Permissions$ device role at any time. Moreover, $teenagers$ can access $Front\_\ Door\_\ Lock$ at any time, whereas they can access $Entertainment\_Devices$ device role when the environment conditions $Teenagers\_\ Entertainment\_Time$ is active. Finally, $parents$ are authorized to access all different device roles at any time.

Next, we consider the dynamic attributes. We introduce one Boolean dynamic user and session attribute $Front\_Door\_Lock\_Token$ which is True when the homeowner has granted this user a token indicating that this user is allowed to access the front door lock device and False otherwise. The detailed mechanism by which the homeowner grant a temporal token for a specific user is outside the scope of $HyBAC_{RC}$ operational model and can be considered as part of an administrative model. We also define three dynamic device attributes: (1) $Device\_Temperature$ whose value is determined by the device's temperature. (2) $UsingStatus$ to indicate that a device is in use (True) or not (False). (3) $UsingUser$, which maps each device to the user currently using it (or is undefined if the device is not in use).

Finally, we define the authorization function, which is a disjunction of six conjunctive propositional clauses (rules) given in Figure 5.3. The first clause gives unconditional access to parents. The following two clauses define teenagers authorization rules with respect to dangerous and non-dangerous kitchen permissions. The fourth, and fifth clauses specify teenagers authorization rules with respect to entertainment devices and front door lock respectively. Finally, the last clause specifies kids access rights with respect to kids friendly content in entertainment devices.

$U = \{alex, bob, anne, suzanne, john\}$
$R = \{kids, parents, teenagers\}$
$UA = \{(bob, parents), (alex, kids), (suzanne, kids), (anne, teenagers), (john, teenagers)\}$
$D = \{Oven, Fridge, FrontDoorLock, PlayStation, TV\}$
$OP = OP_{Oven} \cup OP_{Fridge} \cup OP_{FrontDoorLock} \cup OP_{PlayStation} \cup OP_{TV}$, where
$OP_{Oven} = \{On_{Oven}, \mathit{Off}_{Oven}, Open_{Oven}, Close_{Oven}\}$,
$OP_{Fridge} = \{Open_{Fridge}, Close_{Fridge}, Check\_temperature\}_{Fridge}\}$,
$OP_{FrontDoorLock} = \{Lock_{FrontDoorLock}, Unlock_{FrontDoorLock}\}$
$OP_{PlayStation} = \{On_{PS}, \mathit{Off}_{PS}\}$,
$OP_{TV} = \{On_{TV}, \mathit{Off}_{TV}, G_{TV}, PG_{TV}, R_{TV}\}$
$P = P_{Oven} \cup P_{Fridge} \cup P_{FrontDoorLock} \cup P_{PlayStation} \cup P_{TV}$, where
$P_{Oven} = \{Oven\} \times OP_{Oven}$,
$P_{Fridge} = \{Fridge\} \times OP_{Fridge}$,
$P_{FrontDoorLock} = \{FrontDoorLock\} \times OP_{FrontDoorLock}$,
$P_{PlayStation} = \{PlayStation\} \times \{On, \mathit{Off}\}$,
$P_{TV} = \{TV\} \times \{On, \mathit{Off}, G, PG, R\}$
Let $P_1 = \{Oven\} \times \{On_{Oven}, Open_{Oven}\}$,
$P_2 = \{Oven\} \times \{\mathit{Off}_{Oven}, Close_{Oven}\}$,
$P_3 = \{TV\} \times \{On, \mathit{Off}, G_{TV}\}$,
$DR = \{Dangerous\_Kitchen\_Permissions, Non\_Dangerous\_Kitchen\_Permissions,$
$Front\_Door\_Lock$,$Kids\_Friendly\_Content$, $Entertainment\_Devices\}$
$PDRA = P_1 \times \{Dangerous\_Kitchen\_Permissions\} \cup$
$(P_2 \cup P_{Fridge}) \times \{Non\_Dangerous\_Kitchen\_Permissions\} \cup$
$P_{FrontDoorLock} \times \{Front\_Door\_Lock\} \cup$
$P_{TV} \times \{Entertainment\_Devices\} \cup$
$(P_3 \cup P_{PlayStation}) \times \{Kids\_Friendly\_Content\}$
$EC = \{Parent\_Is\_In\_The\_Kitchen, Weekends, Evenings, Nights, TRUE\}$
$ER = \{Teenagers\_Kitchen\_Time, Kids\_Entertainment\_Time, Teenagers\_Entertainment\_Time, Any\_Time\}$
$EA = \{(\{Parent\_Is\_In\_The\_Kitchen\}, Teenagers\_Kitchen\_Time), (\{weekends, evenings\}, Kids\_Entertainment\_Time),$
$(\{weekends, evenings\}, Teenagers\_Entertainment\_Time), (\{weekends, nights\}, Teenagers\_Entertainment\_Time),$
$(TRUE, Any\_Time)\}$
$RP = \{(teenager, \{Teenagers\_Kitchen\_Time\}), (teenager, \{Teenagers\_Entertainment\_Time\}),$
$(teenagers, \{Any\_Time\}), (kids, \{Kids\_Entertainment\_Time\}),$
$(parents, \{Any\_Time\})\}$
$RPDRA = \{((kids, \{Kids\_Entertainment\_Time\}), Kids\_Friendly\_Contents),$
$((teenager, \{Teenagers\_Entertainment\_Time\}), Entertainment\_Devices),$
$((teenager, \{Teenagers\_Kitchen\_Time\}), Dangerous\_Kitchen\_Permissions),$
$((teenagers, \{Any\_Time\}), Non\_Dangerous\_Kitchen\_Permissions),$
$((teenagers, \{Any\_Time\}), Front\_Door\_Lock),$
$((parents, \{Any\_Time\}), Entertainment\_Devices),$
$((parents, \{Any\_Time\}), Non\_Dangerous\_Kitchen\_Permissions),$
$((parents, \{Any\_Time\}), Dangerous\_Kitchen\_Permissions),$
$((parents, \{Any\_Time\}), Front\_Door\_Lock)\}$

$PRConstraints = \{\{(Oven, On_{Oven}), (Oven, \mathit{Off}_{Oven}), (Fridge, Open_{Fridge}), (Fridge, Close_{Fridge})\}, \{kids\})\}$

$DUSA = \{Front\_Door\_Lock\_Token\}$,
$DDA = \{Device\_Temperature, UsingStatus, UsingUser\}$
$Front\_Door\_Lock\_Token : u : U \rightarrow \{True, False\}$
$Front\_Door\_Lock\_Token : s : S \rightarrow \{True, False\}$
$Device\_Temperature : d : D \rightarrow \{x | x \text{ is an oven temperature}\}$
$UsingStatus : d : D \rightarrow \{True, False\}, UsingUser : d : D \rightarrow U$

Figure 5.2: $HyBAC_{RC}$ Use Case: Basic Configuration

$$
\begin{aligned}
&Authorization(s:S, op:OP, d:D) \equiv \\
&(parents \in R(s)) \vee \\
&(teenager \in R(s) \wedge\ Dangerous\_Kitchen\_Permissions \in drole((op,d)) \wedge \\
&\qquad\qquad Device\_Temperature(d) \leq 250^{\circ}) \vee \\
&(teenager \in R(s) \wedge\ Non\_Dangerous\_Kitchen\_Permissions \in drole((op,d))) \vee \\
&(teenager \in R(s) \wedge\ Entertainment\_Devices \in drole((op,d)) \wedge \\
&\qquad\qquad (\neg UsingStatus(d) \vee UsingUser(d) = user(s)) \vee \\
&(teenager \in R(s) \wedge\ Front\_Door\_Lock \in drole((op,d)) \ \wedge Front\_Door\_Lock\_Token(s) = True) \vee \\
&(kids \in R(s) \ \wedge Kids\_Friendly\_Contents \in drole((op,d)) \wedge \\
&\qquad\qquad (\neg UsingStatus(d) \vee UsingUser(d) = user(s))
\end{aligned}
$$

Figure 5.3: $HyBAC_{RC}$ Use Case: Attributes Authorization Function

5.3 $HyBAC_{AC}$ model

In this section, we introduce the $HyBAC_{AC}$ model. Figure 5.4 conceptually depicts the model components. The model formal definition is given in Table 5.3, and Table 5.4. $HyBAC_{AC}$ model is inspired by the HABAC model [20] introduced in Section 4.5, it fully incorporates HABAC model components while adding two new components: Roles, and Permission-Role constraints. Furthermore, the check access predicate is different from HABAC model's check access predicate. The components of HABAC are shown in blue while the added $HyBAC_{AC}$ components are shown in grey in Figure 5.4.

To develop $HyBAC_{AC}$ we followed an attribute-centric design [70], where the role is just one of many attributes. This is in contrast with conventional RBAC where a role is a collection of permissions.

5.3.1 $HyBAC_{AC}$ Formal definition

The basic components of $HyBAC_{AC}$ are discussed below.

HABAC Basic Sets and Functions:

All HABAC components are incorporated into $HyBAC_{AC}$, as shown in blue in Figure 5.4 with the definitions given in the top portion of Table 5.3. Users(U) are humans interacting directly with smart things. Sessions (S) or subjects are created by users to perform some actions in the system. Environment states (ES) represent the current instant of time environment picture that we want to

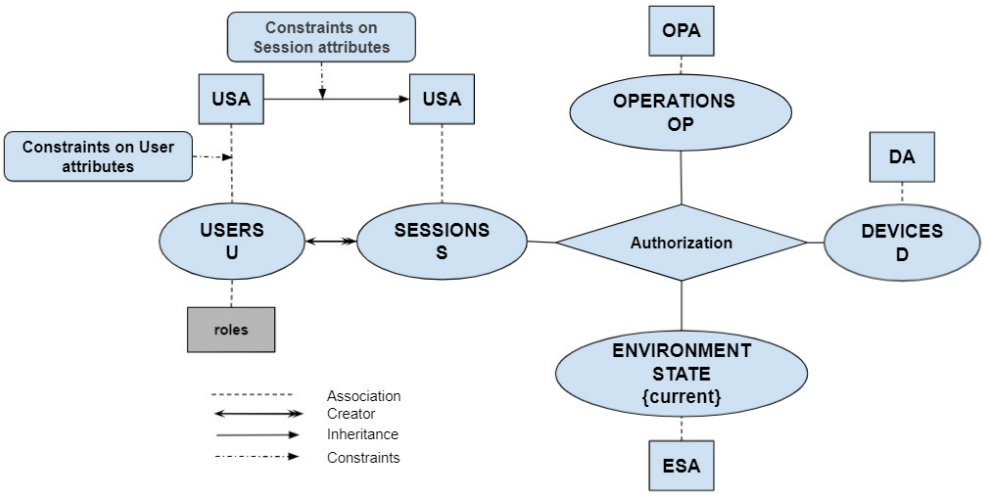

Figure 5.4: Smart Home IoT $HyBAC_{AC}$ Model

describe. Devices (D) are smart home devices. Operations (OP) are actions on devices as specified by device manufacturers. Attributes are characteristics of users/sessions, devices, operations, and environment states which are used in access control decisions. An attribute is a function that takes an entity such as a user and returns a specific value from its range. An atomic valued attribute will return one value from the range, while a set valued attribute will return a subset of the range. We have two types of attributes: static, and dynamic. Operation and device attribute functions are partial functions; we may have some devices or some operations that are not assigned to some attributes. On the other hand, users, sessions, and environment state attributes are total functions.

HABAC Constraints:

A constraint is an invariant that must be maintained at all times. $HyBAC_{AC}$ encompasses HABAC constraint as follows.

i Constraints on user attributes: these constraints enforce restrictions on user attributes.

ii Constraints on session attributes: these constraints enforce restrictions on subject attributes.

Roles(R):

R is a finite set (range) of roles (aka anti-roles) specified by the homeowner. The function $roles$ maps each user to a subset of roles. The set of roles mapped to each user is defined by the homeowner.

Permission-role constraints:

$HyBAC_{AC}$ incorporates constraints that prevent specific roles from accessing specific operations on specific devices. For example, having a permission role constraint $prc_i \in PRConstraints$, where $prc_i = (\{(d_k, op_l)\}, \{r_m\})$ implies that a session s_i created by a user u_j, where $r_m \in roles(u_j)$, can not access the operation op_l on the device d_k. Permission-role constraints are checked during execution time as part of the check access predicate. This is unlike the case of $EGRBAC$ and $HyBAC_{RC}$ where the permission-role constraints are enforced at administration time to prevent prohibited assignments.

Attributes Authorization function:

It is a two-valued boolean function which is evaluated for each access decision. It is defined using the grammar of Table 5.4. 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 $usa(s_i)$, $opa(op_k)$, $da(d_j)$, 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.

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 context of environment state $current$ the $CheckAccess(s_i, op_k, d_j, current)$ predicate evaluates to True if the following three conditions satisfied:

- The operation op_k is assigned to the device d_j by the device manufacturer.
- The authorization function is evaluated to True.
- There is no prc_i in the set of permission-role constraints that prevent this access.

5.3.2 Use Case Demonstration

In this section, we illustrate how to configure $HyBAC_{AC}$ to achieve the same goals of the use case presented in Section 5.2.2. For this purpose $HyBAC_{AC}$ will be configured as shown in

Table 5.3: $HyBAC_{AC}$ Model Formalization Part I: Basic Sets and Components

Basic Sets and Functions

$-U$ is a finite sets of users (home owner specified)

$-S$ is the set of sessions (each session is created, terminated and controlled by an individual user)

$-$The function $user(s) : S \rightarrow U$ maps each session to its unique creator and controlling user

$-D$ is the set of devices deployed in the smart home (home owner deployed)

$-OP$ is the set of possible operations on devices (device manufacturers specified)

$-$The function $ops : D \rightarrow 2^{OP}$ specifies the valid operations for each device (device manufacturers 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 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 $att \in USA$, $att(s) = att(user(s))$ at all time
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

$-attType : USA \cup DA \cup OPA \cup ESA \rightarrow \{set, atomic\}$.

$-$Each $att \in USA \cup DA \cup OPA \cup ESA$ 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:

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

$-$Every $att \in USA \cup DA \cup OPA \cup ESA$, att is designated to be either a static or dynamic attribute where dynamic attributes must have corresponding sensors deployed in the smart home (under home owner control)

$-$Static attribute ranges and values are set and changed by administrator actions (by home owner or device manufacturers)

$-$Dynamic attribute ranges and values automatically determined by sensors deployed in the smart home (under home owner control) or set and changed by home owner.

Constraints

$-UAConstraint \subseteq UAP \times 2^{UAP}$ is the user attribute constraints relation (home owner specified) where

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

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

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

$-SAConstraint \subseteq UAP \times 2^{UAP}$ is the session attribute constraints relation (home owner specified)

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

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

Attributes Authorization Function

$-Authorization(s : S, op : OP, d : D, current : ES)$ is a logic formula defined using the grammar of Table 5.4 (home owner specified)
It is evaluated for a specific session s_i, operation op_k, device d_j and environment state $current$ as specified in Table 5.4

Roles (aka Anti-Roles)

$-R$ is a finite set of roles (aka anti-roles) (home owner specified)

$-$The function $roles : U \rightarrow 2^R$ maps each user to a subset of roles (home owner specified)

$-PRConstraints \subseteq 2^P \times 2^R$, many to many permission-role constraints relation (home owner specified)
where $P \subseteq D \times OP$ is a derived relation such that $(d, op) \in P \Leftrightarrow op \in ops(d)$

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:
$op_k \in ops(d_j) \ \wedge \ Authorization(s_i, op_k, d_j, current)) \wedge$
$(\forall (P_x, R_y) \in PRConstraints)[((op_k, d_j) \notin P_x) \vee (roles(user(s_i))) \cap R_y = \phi]$

$U = \{alex, bob, anne, suzanne, john\}$
$D = \{Oven, Fridge, FrontDoorLock, TV, PlayStation\}$
$OP = OP_{Oven} \cup OP_{Fridge} \cup OP_{FrontDoorLock} \cup OP_{PlayStation} \cup OP_{TV}$, where
$OP_{Oven} = \{On_{oven}, \mathit{Off}_{oven}, Open_{oven}, Close_{oven}\}$,
$OP_{Fridge} = \{Open_{fridge}, Close_{fridge}, CheckTemperature_{fridge}\}$,
$OP_{FrontDoorLock} = \{Lock_{FrontDoorLock}, Unlock_{FrontDoorLock}\}$,
$OP_{PlayStation} = \{On_{PS}, \mathit{Off}_{PS}\}$
$OP_{TV} = \{On_{TV}, \mathit{Off}_{TV}, G_{TV}, PG_{TV}, R_{TV}\}$
$ops(Oven) = OP_{Oven}, ops(Fridge) = OP_{Fridge}, ops(FrontDoorLock) = OP_{FrontDoorLock}, ops(PlayStation) = OP_{PlayStation}, ops(TV) = OP_{TV}$

$USA = \{FamilyRole, FrontDoorLockToken\}$
$FamilyRole : U \rightarrow \{parent, kid, teenager\}$
$FamilyRole(alex) = FamilyRole(suzanne) = kid$
$FamilyRole(anne) = FamilyRole(john) = teenager$
$FamilyRole(bob) = parent$
$FrontDoorLockToken : U \rightarrow \{True, False\}$
This attribute is a dynamic attribute that is dynamically set by home owner

$DA = \{DangerouseKitchenDevices, FrontDoorLockDevice, EntertainmentDevices, DeviceTemperature, UsingStatus, UsingUser\}$
$DangerouseKitchenDevices : D \rightarrow \{True, False\}$
$DangerouseKitchenDevices(Oven) = True$, $DangerouseKitchenDevices(Fridge) = False$, all other values are undefined
$FrontDoorLockDevice : D \rightarrow \{True, False\}$
$FrontDoorLockDevice(FrontDoorLock) = True$, all other values are undefined
$EntertainmentDevices : D \rightarrow \{True, False\}$
$EntertainmentDevices(TV) = EntertainmentDevices(PlayStation) = True$, all other values are undefined
$DeviceTemperature : D \rightarrow \{x | x \text{ is an oven temperature}\}$
$DeviceTemperature(Oven)$ is dynamically set by sensors, all other values are undefined
$UsingStatus : D \rightarrow \{True, False\}$
$UsingStatus(TV)$ and $UsingStatus(PlayStation)$ are dynamically set by sensors, all other values are undefined
$UsingUser : D \rightarrow U$
$UsingUser(TV)$ and $UsingUser(PlayStation)$ are s dynamically set by sensors, all other values are undefined

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

$OPA = \{KidsFriendlyContent, DangerouseKitchenOperation\}$
$KidsFriendlyContent : OP \rightarrow \{True, False\}$
$KidsFriendlyContent(G_{TV}) = KidsFriendlyContent(On_{TV}) = KidsFriendlyContent(\mathit{Off}_{TV}) = KidsFriendlyContent(On_{PS}) = KidsFriendlyContent(\mathit{Off}_{PS}) = True$
$KidsFriendlyContent(PG_{TV}) = KidsFriendlyContent(R_{TV}) = False$
All other values are undefined
$DangerouseKitchenOperation : OP \rightarrow \{True, False\}$
$DangerouseKitchenOperation(On_{Oven}) = DangerouseKitchenOperation(Open_{Oven}) = True$
$DangerouseKitchenOperation(\mathit{Off}_{Oven}) = DangerouseKitchenOperation(Close_{Oven}) = False$
$DangerouseKitchenOperation(Open_{Fridge}) = DangerouseKitchenOperation(Close_{Fridge}) = DangerouseKitchenOperation(CheckTemperature_{Fridge}) = False$
All other values are undefined

$R = \{kid\}$
$roles(alex) = roles(suzanne) = \{kid\}$
$roles(anne) = roles(john) = roles(bob) = \emptyset$
$PRConstraints = \{(\{(Oven, On_{Oven}), (Oven, \mathit{Off}_{Oven}), (Fridge, Open_{Fridge}), (Fridge, Close_{Fridge})\}, \{kid\})\}$

Figure 5.5: Use Case Configuration in $HyBAC_{AC}$: Basic Configuration

Table 5.4: $HyBAC_{AC}$ Model Formalization Part II: Attributes Authorization Function

Attributes Authorization Function

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

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

$$
\begin{aligned}
&Authorization(s : S, op : OP, d : D, current : ES) \equiv \\
&(parent \in FamilyRole(s)) \vee \\
&(teenager \in FamilyRole(s) \wedge ParentInKitchen(current) \wedge DangerouseKitchenDevices(d) \wedge \\
&DangerouseKitchenOperation(op) \wedge \\
&\qquad Device\_Temperature(d) \leq 250^{\circ}) \vee \\
&(teenager \in FamilyRole(s) \wedge \neg DangerouseKitchenOperation(op)) \vee \\
&(teenager \in FamilyRole(s) \wedge FrontDoorLockDevice(d) \wedge FrontDoorLockToken(s)) \vee \\
&(teenager \in FamilyRole(s) \wedge day(current) \in \{Sa, S\} \wedge 17{:}00 \leq time(current) \leq 23{:}59) \wedge EntertainmentDevices(d) \wedge \\
&\qquad (\neg UsingStatus(d) \vee UsingUser(d) = user(s)) \vee \\
&(kid \in FamilyRole(s) \wedge day(current) \in \{Sa, S\} \wedge 17{:}00 \leq time(current) \leq 19{:}00) \wedge EntertainmentDevices(d) \wedge \\
&\qquad KidsFriendlyContent(op) \wedge (\neg UsingStatus(d) \vee UsingUser(d) = user(s))
\end{aligned}
$$

Figure 5.6: Use Case Configuration in $HyBAC_{AC}$: Authorization Function

Figure 5.5. We have five users bob, $alex$, $suzanne$, $john$, and $anne$. We have five devices $Oven, Fridge, FrontDoorLock, PlayStation$ and TV. These devices are assigned by the house owner to different device attributes as follows. $Fridge \leftarrow (DangerouseKitchenDevices : False)$, $Oven \leftarrow (DangerouseKitchenDevices : True)$, $FrontDoorLock \leftarrow (FrontDoorLockDevice : True)$, $PlayStation \leftarrow (EntertainmentDevices : True)$, and $TV \leftarrow (EntertainmentDevices : True)$. Similarly, different operations are assigned to different operation attributes. Since the operation, and device attribute functions are partial functions; we may have some devices or some operations that are not assigned to some attributes.

We have two user/session attribute functions $FamilyRole$, and $FrontDoorLock$. $FamilyRole$ defines the user role in the family and has a range of three values $\{parent, kid, teenager\}$. The

family role function value for $bob, alex, suzanne, john,$ and $anne$ is set to the value $\{parent\}$, $\{kids\}$, $\{kids\}$, $\{teenagers\}$, and $\{teenagers\}$ respectively. We have one environment state $current$ which has three attribute functions (day, $time$, and $ParentInKitchen$).

The authorization function set $Authorization(s : S, op : OP, d : D, current : ES)$ shown in Figure5.6 is a disjunction of six propositional statements. The first statement gives parents access to anything unconditionally. The second statement authorizes teenagers to use dangerous kitchen operations on dangerous kitchen devices only when one of the parents is in the kitchen and the device temperature is below $250°$. The third statement gives teenagers access to non dangerous kitchen operations unconditionally. The fourth statement allows a teenager to use the front door lock device only if he/she is granted the front door lock token by the parent. The fifth statement gives teenagers access to entertainment devices during a specific time, and if the requested device is not in use by another user. Finally, the sixth statement permits kids to use kids friendly operations on entertainment devices during a specific time, and if they are currently not in use by another user.

5.4 Implementation

5.4.1 Enforcement Architecture

In this section, we present proof of concept demonstrations of $HyBAC_{RC}$, and $HyBAC_{AC}$ models by enforcing the use cases presented in Section 5.2.2, and Section 5.3.2 respectively using AWS (Amazon Web Services) IoT service [4] to confirm the applicability of our model using commercially available systems. We have used simulations to reflect real smart home devices, however, this does not undermine the plausibility, use, and advantage of our proposed model as further elaborated in the following discussion.

We adopted the smart home IoT architecture shown in Figure 5.7 which was first introduced by Geneiatakis et al [51]. In this architecture, the IoT devices are connected to a corresponding hub and are not directly accessed by other devices or by users. In general, there are two types of access. In local access, users directly interact with the IoT devices through the connectivity services

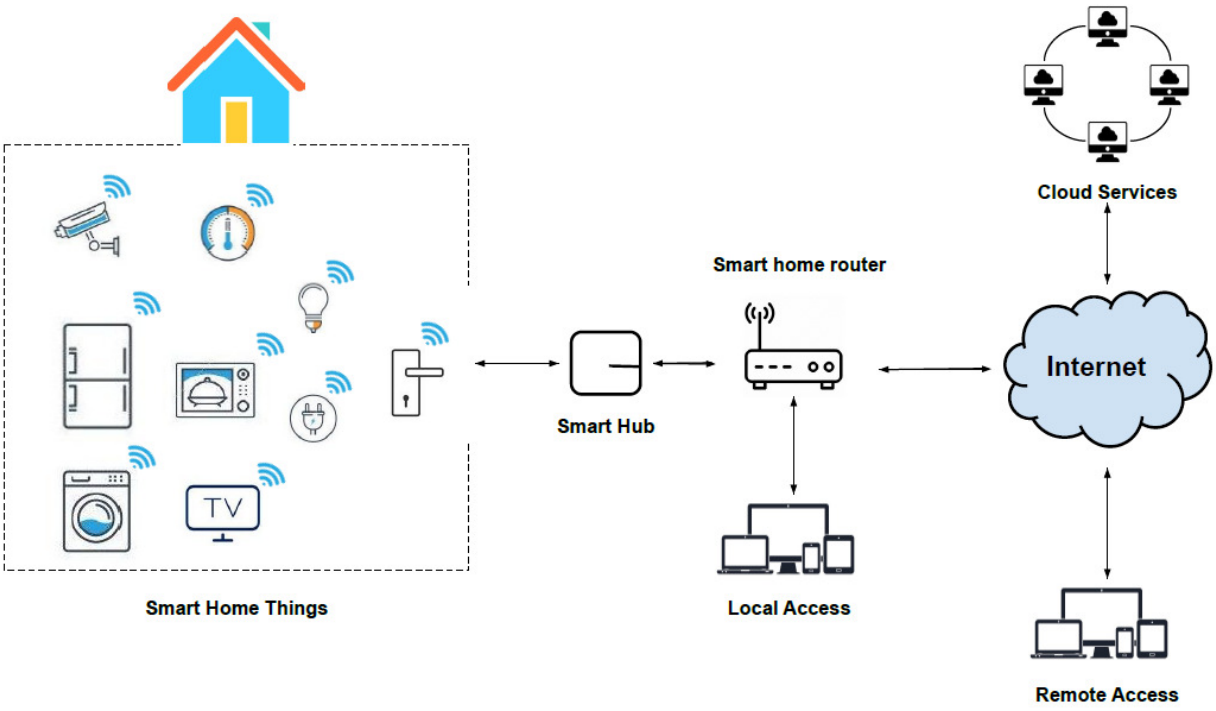

Figure 5.7: Enforcement Architecture (adapted from [51])

provided by the hub. In remote access, users access IoT devices via cloud services, which in turn communicate with the smart hub via the Internet to access these devices. In our enforcement, we only handled local communication.

First, we created an AWS account, then we configured and deployed Greengrass [6]. The Greengrass SDK (Software Development Kit) extends cloud capabilities to the edge, which in our case is the smart home. It serves as a smart hub and a policy engine. It enables devices to process data closer to the source of information, and communicate securely on local networks. We deployed Greengrass on a dedicated virtual machine with 1 virtual CPU and 2 GB of RAM running ubuntu server 18.04.5 LTS. Second, we simulated the five users (the devices which are used by users to access the smart things, for example their phones), and the five devices (The smart things that users want to access) of the two use cases using AWS IoT device SDK for Python [5] provided by AWS on different virtual machines. Each machine has 1 virtual CPU and 2 GB of RAM running ubuntu server 18.04.5 LTS. Third, through AWS IoT management console we created one virtual object (digital shadow) for each physical device (smart thing devices need to be accessed

or user's access device). Each physical device and its corresponding shadow are cryptographically linked via digital certificates with attached authorization policies. MQTT protocol [11] is used by the devices and users to communicate to the AWS IoT service with TLS security [2]. MQTT standard is a machine-to-machine (M2M) lightweight publish/subscribe messaging protocol, specially designed for constrained devices. Each shadow has a set of predefined MQTT topics/channels to allow its interaction with other IoT devices and applications.

$HyBAC_{RC}$ Enforcement

Here, we created two Json files: (a) UsersRolesAssignment.jason. This defines the assignments of users to different roles. (b) ModelComponentConfiguration.json. This file defines and configures the rest of $HyBAC_{RC}$ components to express our use case. Moreover, we utilized the lambda function service in AWS IoT platform [7] to receive different requests of users to access the smart devices in the house, analyze each request according to the content of our json files, and finally trigger the desired actions on the corresponding simulated devices. Code is written in Python 3.7 and running on a long-lived lambda function with 500 MB Memory Limit, 30 seconds timeout. The lambda function, the UsersRolesAssignment.json file, and the ModelComponentConfiguration.json file are all configured in the Greengrass.

$HyBAC_{AC}$ Enforcement

To enforce $HyBAC_{AC}$, we created four json files as following, UsersAttributes.json, DevicesAttributes.json, OperationAttributes.json, and EnvironmnetAttributes.json to captures different users 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. Similar to $HyBAC_{RC}$ enforcement, here we also utilized AWS IoT platform lambda function to receive different users requests, analyze those requests according to the contents of the json files, allow or deny the requested accesses, and finally trigger the desired actions on the corresponding devices. Code is written in Python 3.7 and running on a long-lived lambda function with 500 MB

Memory Limit, 30 seconds timeout.

Local Communication handling

Figure 5.8 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, when the user tries to send permission request to unlock the front door lock through his mobile phone while he is inside the house. First, a request is sent via MQTT protocol to the virtual object (or local shadow) corresponding to the user phone in Greengrass through the publish/subscribe relation between the user's phone, and the local shadow. The local shadow gets notified with the request, and sends it to the lambda function through MQTT publish/subscribe protocol. After that, the lambda function analyzes the request according to the model implemented. In the case of $HyBAC_{RC}$, it analyzes the request according to the UsersRolesAssignment.json and ModelComponentConfiguration.json files and makes a decision whether to authorize the user to unlock the front door or not. On the other hand, in the case of $HyBAC_{RC}$, it analyzes the request according to the content of UsersAttributes.json, DevicesAttribu tes.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, the local shadow gets notified and updates the user's phone that the permission was denied. The front door lock in this case does not get an indication that a user attempted to access it. If the request is granted, the front door lock local shadow is notified through its update topic and updates the front door lock with the unlock command. After the front door lock is unlocked, it notifies its shadow through publishing to the shadow update topic. The front door lock local shadow then notifies the lambda function which in turn notifies the user phone's local shadow. Finally, the user phone's local shadow updates the user's phone that the TV was turned on successfully.

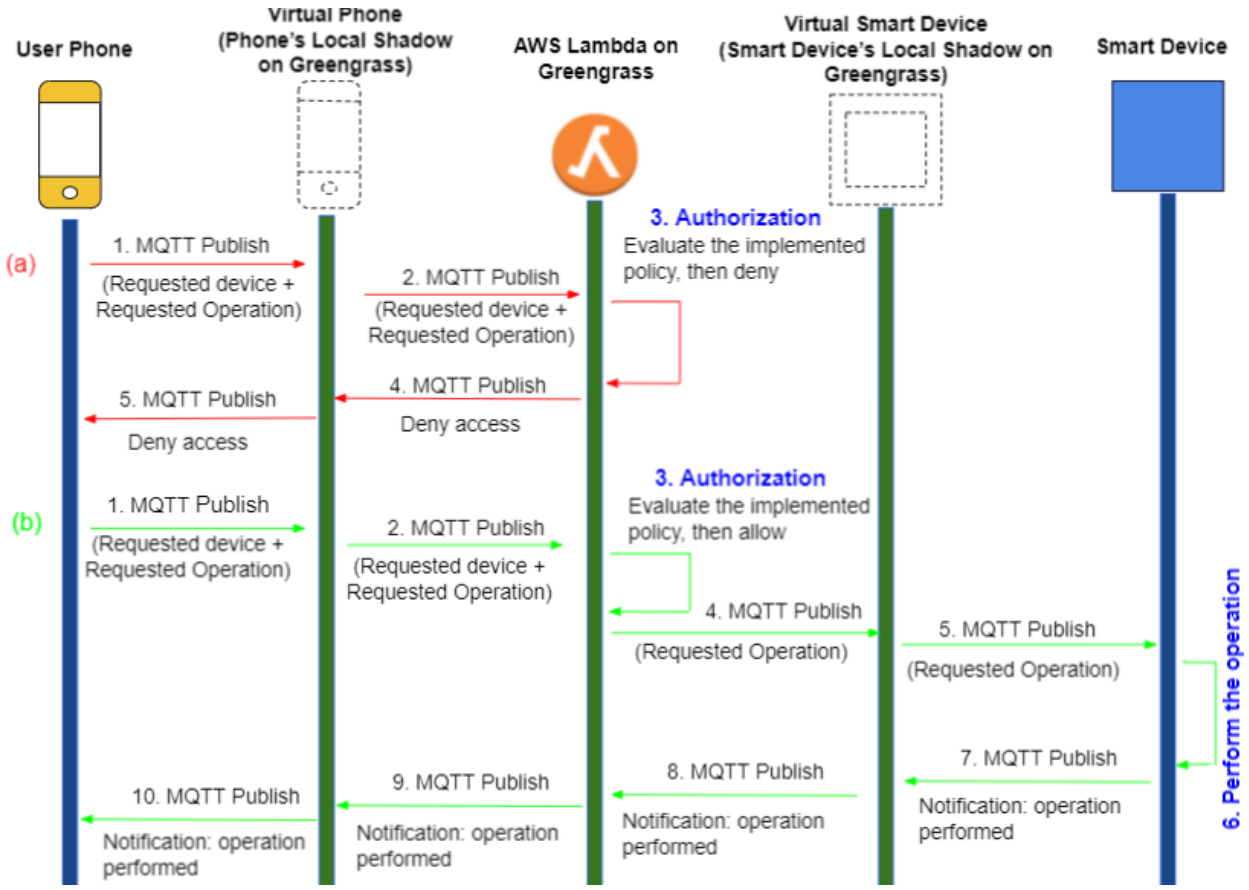

Figure 5.8: Local Request Handling in Our System

5.4.2 Performance Results

We executed multiple test scenarios to check the policy machine's response in each case. Furthermore, we analyzed the performance of our implementation. In particular, we measured the average lambda function execution time under different conditions. We implemented the configuration of the same use case given in Section 5.2.2, and Section 5.3.2 for $HyBAC_{RC}$ and $HyBAC_{AC}$ models respectively. We analyzed three different cases with three different loads of requests, each unique load of requests was executed 10 times to measure the average lambda processing time as follows. (a) When one user is sending requests to multiple devices at the same time. Table 5.5 shows the measured average lambda function execution time in this test case for $HyBAC_{RC}$ and $HyBAC_{AC}$ implementation. The first, second, and third rows show the average time when the parent Bob requests to lock the front door lock, the average time when Bob requests to lock the front door

lock, turn on the TV, and turn on the PlayStation at the same time, and the average time when Bob requests to lock the front door lock, open the fridge, turn on the oven, the TV, and the PlayStation at the same time respectively. All the requests were approved as they were supposed to according to our configured policies. (b) When multiple users are sending requests to multiple devices at the same time (one user per device). Table 5.6 describes the measured average lambda function execution time in this test case for $HyBAC_{RC}$ and $HyBAC_{AC}$ implementation. The first, second, and third rows show the average time when the parent Bob requests to lock the front door lock, the average 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 at the same time, the average time when the three access requests tested in the second row are carried again in addition to, the kid Suzanne requests to turn on the TV, and the teenager John requests to open the oven while one of the parent is in the kitchen and the oven temperature is 100°. The two systems responded correctly where all the requests were granted except for when the kid Alex was trying to turn on the oven since according to our configuration he is not allowed to, and when Suzzane was trying to turn on the TV since the testing was performed during a weekday, and according to our configuration kids are not allowed to access TV during weekdays. (c)Finally, Table 5.7 illustrates the average lambda function execution time when multiple users are sending requests to one device at the same time in $HyBAC_{RC}$ and $HyBAC_{AC}$ implementation. 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. The two systems responded correctly where all the requests were denied except for when the parent Bob requests to lock the front door lock.

From the tables, we can notice that the two models are functional and applicable using commercially available technology. Furthermore, the captured average lambda processing times are generally low, and the captured lambda processing time in $HyBAC_{AC}$ always less than those in $HyBAC_{RC}$.

We understand that practical smart homes will have different and more complicated scenarios.

Table 5.5: One User Sending Requests to Multiple Devices

Users	Devices	$HyBAC_{RC}$ L.P.T	$HyBAC_{AC}$ L.P.T	N.R
1	1	1.8343	1.2661	10
1	3	1.7408	1.3118	30
1	5	1.76588	1.3503	50

Table 5.6: Multiple Concurrent Instances of One User Sending Request to One Device.

Users	Devices	$HyBAC_{RC}$ L.P.T	$HyBAC_{AC}$ L.P.T	N.R
1	1	1.8343	1.2661	10
3	3	1.8385	1.3803	30
5	5	2.01128	1.3247	50

Table 5.7: Multiple Users Sending Requests to One Device

Users	Devices	$HyBAC_{RC}$ L.P.T	$HyBAC_{AC}$ L.P.T	N.R
1	1	1.8343	1.2661	10
3	1	1.73177	1.2818	30
5	1	1.8771	1.2654	50

L.P.T $\equiv$ Lambda function processing time in ms.
N.R $\equiv$ Total number of requests (10 per unique request)

Although a detailed performance evaluation is eventually necessary by simulating a large set of smart things, we believe that our proof of concept implementation in AWS is to showcase the practical viability and use of fine grained security policies in the context of smart home IoT, without the need to capture a large set of scenarios from the real world. Such a scaled setting will not reflect any change in security policy evaluation. We consider a more detailed performance analysis as an extension of this work.

5.5 Constructing $HyBAC_{AC}$ FROM $HyBAC_{RC}$

In this section, we take a further step toward comparing $HyBAC_{RC}$ (a role-centric hybrid model), and $HyBAC_{AC}$ (an attribute-centric hybrid model). We introduce $HyBAC_{AC}$ configuration that translates $HyBAC_{RC}$ policies in a manner that they can be implemented by $HyBAC_{AC}$. The purpose is to see whether we can fully express any $HyBAC_{RC}$ configuration in $HyBAC_{AC}$ model, and if not which model is more expressive, and in what terms. Here, we followed a similar approach to the one introduced in Section 4.3 for constructing $HABAC$ from $EGRBAC$.

5.5.1 Approach

Step 1: Convert the $HyBAC_{RC}$ dynamic attributes authorization function $Authorization(s : S, op : OP, d : D)$ into a disjunctive normal form (DNF). All logical formulas can be converted into an equivalent DNF form. Figure 5.9 shows the attribute authorization function of our $HyBAC_{RC}$ use case which is introduced in Figure 5.3 after following the standard approach to convert it into a DNF format. We call each conjuncted term a condition. We have role conditions (conditions that involve users roles), dynamic user attributes conditions (involve dynamic user attributes), device role conditions (involve device roles), dynamic device attributes conditions (involve dynamic device attributes), and mix conditions (involve two types of attributes).

Here, we define a function that takes a user role $r_i \in R$ and a device role $dr_j \in DR$ as inputs, loops through each clause in the DNF form of the $Authorization(s : S, op : OP, d : D)_{RC}$, and returns set of sets of dynamic user attributes conditions, dynamic device attributes conditions, and

Table 5.8: $HyBAC_{RC}$ Configuration in $HyBAC_{AC}$

- $U_{AC} = U_{RC}$
- $USA_{AC} = \{Relationship\}$
- $Range(Relationship) = R_{RC}$
- $Relationship : u \in U_{AC} \rightarrow 2^{R_{RC}}$
- $Relationship : s \in S_{AC} \rightarrow 2^{R_{RC}}$
- $(\forall u_i \in U_{AC})[Relationship(u_i) = \{r_x | (u_i, r_x) \in UA_{RC}\}]$
- $USA_{AC} = USA_{AC} \cup DUSA_{RC}$
- $(\forall usa \in USA_{AC} \cap DUSA_{RC})[Range(usa)$ in $HyBAC_{AC}$ = $Range(usa)$ in $HyBAC_{RC}]$
- $(\forall usa \in USA_{AC} \cap DUSA_{RC})(\forall u \in U_{AC})[usa(u)$ in $HyBAC_{AC} = usa(u)$ in $HyBAC_{RC}]$

- $UAConstraint_{AC} = \{uac_i\}$
- For all $ssdc_i = (r_i, R_j) \in SSDConstraints_{RC}$:
$uac_i = ((Relationship, r_i), UAP_j)$, where $UAP_j = \{(Relationship, r_n) | r_n \in R_j\}$

- $SAConstrain_{AC}t = \{sac_i\}$
- For all $dsdc_i = (r_i, R_j) \in DSDConstraints_{RC}$:
$sac_i = ((Relationship, r_i), SAP_j)$, where $SAP_j = \{(Relationship, r_n) | r_n \in R_j\}$

- $ES_{AC} = \{Current\}$
- $ESA_{AC} = ER_{RC}$
$(\forall esa_i \in ESA_{AC})[esa_i : es \in ES_{AC} \rightarrow \{True, False\}]$

- $D_{AC} = D_{RC}, OP_{AC} = OP_{RC}$
- $DA_{AC} = DR_{RC} \cup DDA_{RC}$
- $OPA_{AC} = DR_{RC}$
- $(\forall att \in DA_{AC} \cap DDA_{RC})[Range(att)\text{in}HyBA_{AC} = Range(att)\text{in}HyBA_{RC}]$
- $(\forall da \in DA_{AC} \cap DDA_{RC})(\forall d \in D_{AC})[da(d)$ in $HyBAC_{AC} = da(d)$ in $HyBAC_{RC}]$
- $(\forall da \in DA_{AC} \cap DR_{RC})$ $[da : d \in D_{AC} \rightarrow \{True, False\}]$
- $(\forall opa_i \in OPA_{AC})[opa : op \in OP_{AC} \rightarrow \{True, False\}]$
- $(\forall (dr_y \in DR_{RC}, p_x \in \{p_i | (p_i, dr_y) \in PDRA_{RC}\}))[dr_y(p_x.op) = True, dr_y(p_x.d) = True]$

- $R_{AC} = R_{RC}$
- $(\forall u_i \in U_{AC})[roles(u_i) = \{r_x | (u_i, r_x) \in UA_{RC}\}]$
-$PRConstraints_{AC} = PRConstraints_{RC}$.

- Initialize the authorization function $Authorization(s : S_{AC}, op : OP_{AC}, d : D_{AC}, current : ES_{AC})$
- For each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA_{RC}$, we construct an authorization policy as following:

1. $SetOfESA =$ "$TRUE$". $\forall (esa \in ER_i)[SetOfESA = SetOfESA +$ "$\wedge$" $+$ "$esa(current) = True$"].
2. $EGRBACPart =$ "r_i" $+$ " $\in$ " $+$ "$Relationship(s)$" $+$ " $\wedge$ " $+$ "$dr_i(op) = True$" $+$ " $\wedge$ " $+$ "$dr_i(d) = True$" $+$ " $\wedge$ " $+$ "$SetOfESA$".
3. $SetOfC = GetRelatedDynamicCondition(r_i, dr_j)$
4. Loop through each $C \in SetOfC$ and do the following:
   (a) Initialize the string $TempAuth$ to equal $EGRBACPart$.
   (b) For every dynamic attribute condition $c_i \in C$, $TempAuth \leftarrow TempAuth +$ " $\wedge$ " $+$ "c_i"
   (c) $Authorization(s : S_{AC}, op : OP_{AC}, d : D_{AC}, current : ES_{AC}) \leftarrow Authorization(s : S_{AC}, op : OP_{AC}, d : D_{AC}, current : ES_{AC}) +$ " $\vee$ " $+$ "$TempAuth$

$$
\begin{aligned}
&Authorization(s:S, op:OP, d:D) \equiv \\
&(parents \in R(s)) \vee \\
&(teenager \in R(s) \wedge\ Dangerous\_Kitchen\_Permissions \in drole((op,d)) \wedge \\
&\qquad\qquad Device\_Temperature(d) \leq 250°) \vee \\
&(teenager \in R(s) \wedge\ Non\_Dangerous\_Kitchen\_Permissions \in drole((op,d))) \vee \\
&(teenager \in R(s) \wedge\ Entertainment\_Devices \in drole((op,d)) \wedge \\
&\qquad\qquad UsingUser(d) = user(s) \vee \\
&(teenager \in R(s) \wedge\ Entertainment\_Devices \in drole((op,d)) \wedge \\
&\qquad\qquad \neg UsingStatus(d) \vee \\
&(teenager \in R(s) \wedge\ Front\_Door\_Lock \in drole((op,d)) \wedge Front\_Door\_Lock\_Token(s) = True) \\
&\vee \\
&(kids \in R(s)\ \wedge Kids\_Friendly\_Contents \in drole((op,d)) \wedge \\
&\qquad\qquad \neg UsingStatus(d) \vee \\
&(kids \in R(s)\ \wedge Kids\_Friendly\_Contents \in drole((op,d)) \wedge \\
&\qquad\qquad UsingUser(d) = user(s)
\end{aligned}
$$

Figure 5.9: The Attribute Authorization Function of The Use Case Described in in Figure 5.3 in DNF Format

mix conditions that need to be satisfied together to enable the access of the role r_i to the device role dr_j according to the attributes authorization function of the $HyBAC_{RC}$ model. For instance:

$$
\begin{aligned}
&GetRelatedDynamicCondition(teenager, Front\_Door\_Lock) = \\
&\{\{Front\_Door\_Lock\_Token(s) = True\}\}. \\
&GetRelatedDynamicCondition(teenager, Entertainment\_Devices) = \\
&\{\{\neg UsingStatus(d)\}, \{UsingUser(d) = user(s)\}\}.
\end{aligned}
$$

Step 2: Follow the $HyBAC_{RC}$ configuration in $HyBAC_{AC}$ approach introduced in Section 5.5.2

5.5.2 $HyBAC_{RC}$ configuration in $HyBAC_{AC}$

The configuration is shown in Table 5.8. In this configuration, for differentiation purposes every $HyBAC_{RC}$ component is followed with the suffix $_{RC}$, similarly, every $HyBAC_{AC}$ component is followed with the suffix $_{AC}$.

The goal is to discover $HyBAC_{AC}$ elements $(U_{AC}, USA_{AC}, ES_{AC}, ESA_{AC}, D_{AC},\ DA_{AC}, OP_{AC}, OPA_{AC})$ and the attribute authorization function set $Authorization(s : S_{AC}, op : OP_{AC}, d : D_{AC}, current\ :\ ES_{AC})_{AC}$ from $HyBAC_{RC}$ policy in such a way that the authorizations are the same as those under $HyBAC_{RC}$. The inputs are $HyBAC_{RC}$ component sets R_{RC}, U_{RC}, $UA_{RC}, DUSA_{RC}$, EC_{RC}, ER_{RC}, EA_{RC}, P_{RC}, D_{RC}, OP_{RC}, DR_{RC}, DDA_{RC} $PDRA_{RC}$, $RPDRA_{RC}$, and $Authorization(s : S, op : OP, d : D)_{RC}$. The steps are following:

The set of users, devices, and operations are the same in both systems. in $HyBAC_{RC}$, user static attributes are expressed through the component roles, while user dynamic attributes are expressed through the component dynamic user/session attribute $DUSA$. Hence, in this configuration, roles are expressed through the user/session attribute $Relationship$ in $HyBAC_{AC}$. $Relationship$ is a user/session attribute that takes a user or a session as an input and returns the set of roles assigned to that user or that session. On the other hand, dynamic user attributes in $HyBAC_{RC}$ are translated into user/session attributes in $HyBAC_{AC}$.

Static separation of duty constraints $SSDConstraints$ are translated into user attributes constraints in $HyBAC_{AC}$. Dynamic separation of duty constraints $DSDConstraints$ are translated into session attributes constraints in $HyBAC_{AC}$.

Environment roles are translated into atomic environment state attributes. Each environment state attribute has a range of values equal to $\{True, False\}$. How to trigger different environment states attributes in response to the environment's changes is outside the scope of this model.

While device roles are ways of categorizing permissions according to static characteristics in $HyBAC_{RC}$, dynamic device attributes are means of expressing dynamic device characteristics. On the other hand, in $HyBAC_{AC}$ device attributes capture both static and dynamic devices characteristics. Hence, device roles DR_{RC} and dynamic device attributes DDA_{RC} in $HyBAC_{RC}$ are both translated into device attributes DA_{AC} in $HyBAC_{AC}$.

Since we do not have permission attributes in $HyBAC_{AC}$, and since a permission is a mapping between a device and an operation, we translate device roles in $HyBAC_{RC}$ into atomic operation attributes and atomic device attributes. For each device role dr_y, we create a device attribute $da_y = dr_y$, and an operation attribute $opa_y = dr_y$. These created attributes have a range of values equal to $\{True, False\}$. Then, for every permission p_x assigned to the device role dr_y in $HyBAC_{RC}$, $dr_y(p_x.op) = True$ and $dr_y(p_x.d) = True$. Permission-role constraints translation is straightforward.

The final step is to construct the authorization function. In $HyBAC_{RC}$ it is the $RPDRA_{RC}$ that gives specific role pairs and hence users access to specific device roles and hence permissions. Fur-

thermore, those permissions granted by $RPDRA_{RC}$ are further constrained by $Authorization(s : S, op : OP, d : D)_{RC}$ according to different dynamic user attributes, and dynamic device attributes. Therefore, to construct the $Authorization(s : S, op : OP, d : D)_{AC}$ we first initialize it, and then loop through each $rpdra_i = ((r_i, ER_i), dr_i) \in RPDRA_{RC}$ and execute the following steps:

1. We construct the string $SetOfESA$ by translating ER_i into a string of logically conjuncted environment state attributes conditions.

2. We construct the string $EGRBACPart$ by extracting users and device roles associated with $rpdra_i$. Then, we translate them into $HyBAC_{AC}$ attribute functions conditions, and logically and them with the string $SetOfESA$ to construct the part of the authorization function corresponding to $rpdra_i$.

3. Get $SetOfC$ the set of sets of the dynamic session attributes conditions, and the dynamic device attributes conditions that further constraints the access of the session assigned to the role r_i to the permissions assigned to the device role dr_i by executing the function $GetRelatedDynamicCondition(r_i, dr_j)$ on the $Authorization(s : S, op : OP, d : D)_{RC}$ after convert it to the DNF format.

4. Loop through each set $C \in SetOfC$, and do the following:

   a Initialize the variable $TempAuth$ to equal to the string $EGRBACPart$ that was constructed earlier.

   b Logically and each $c_i \in C$ to the authorization policy $TempAuth$.

   c Finally, we add (logically or) the constructed authorization policy $TempAuth$ to the final authorization function $Authorization(s : S, op : OP, d : D)_{AC}$.

5.6 Constructing $HyBAC_{RC}$ from $HyBAC_{AC}$

In this section, we introduce our methodology to construct $HyBAC_{RC}$ components and configurations from $HyBAC_{AC}$ policy configuration. In this section, for differentiation purposes every

$HyBAC_{RC}$ component is followed with the suffix $_{RC}$, similarly, every $HyBAC_{AC}$ component is followed with the suffix $_{AC}$. Here, we follow the same approach used to construct $EGRBAC$ from $HABAC$ which introduced in Section 4.4 with some few modifications as shown in the following.

5.6.1 Smart Home Use Case Description

In this section we demonstrate a smart home use case configuration in $HyBAC_{AC}$ system. This use case has the following goals: (a) Authorize teenagers to use dangerous kitchen devices (oven) only when a parent is in the kitchen and the current temperature of the oven is less than or equal to $250°$. (b) Authorize teenagers to use non dangerous kitchen devices (fridge) unconditionally. (d) Authorize teenagers to use the front door lock device (lock, and unlock) if they are currently temporary granted that access permission by one of the parents. (e) Allow teenagers to use entertainment devices unconditionally. (f) Allow kids to use kids friendly operations and contents in entertainment devices which are, turning on and off the device, and G content during weekends between 12:00 pm and 7:00 pm, or weekdays between 5:00 pm and 7:00 pm only. (g) Finally, allow parents to use any operation in any device unconditionally. Figure 5.10 illustrates this use case configuration in $HyBAC_{AC}$. In the following subsections we construct an $HyBAC_{RC}$ configuration from it is equivalent $HyBAC_{AC}$ configuration provided in Figure 5.10.

5.6.2 From Authorization policy to Authorization Array

In $HyBAC_{AC}$, authorization function is a logical clause. First we convert it into a disjunctive normal form (DNF). Figure 5.11 shows the authorization set of our smart home use case which introduced in Figure 5.10 after following the standard approach to convert it into a DNF format.

We call each conjuncted term a condition. We have session, environment, device, operation, and mix conditions which are conditions that involve user/session, environment, device, operation, and any two types of attributes respectively. Each condition is either static condition or dynamic condition. Static condition correspond to static attribute functions. Dynamic conditions correspond to dynamic attribute functions. In the DNF authorization function of our smart home use case

$U = \{alex, bob, anne\}$
$D = \{Oven, Fridge, FrontDoorLock, TV\}$
$OP = OP_{Oven} \cup OP_{Fridge} \cup OP_{FrontDoorLock} \cup OP_{TV}$, where
$OP_{Oven} = \{On_{oven}, \mathit{Off}_{oven}\}$,
$OP_{Fridge} = \{Open_{fridge}, Close_{fridge}\}$,
$OP_{FrontDoorLock} = \{Lock, Unlock\}$,
$OP_{TV} = \{On_{TV}, \mathit{Off}_{TV}, G, PG, R\}$
$ops(Oven) = OP_{Oven}, ops(Fridge) = OP_{Fridge}, ops(FrontDoorLock) = OP_{FrontDoorLock}, ops(TV) = OP_{TV}$

$USA = \{FamilyRole, FrontDoorLockToken\}$
$FamilyRole : U \rightarrow \{parent, kid, teenager\}$
$FamilyRole(alex) = \{kid\}$
$FamilyRole(anne) = \{teenager\}$
$FamilyRole(bob) = \{parent\}$
$FrontDoorLockToken : U \rightarrow \{True, False\}$
This attribute is a dynamic attribute that is dynamically set by home owner

$DA = \{DangerouseKitchenDevices, FrontDoorLockDevice, DeviceTemperature\}$
$DangerouseKitchenDevices : D \rightarrow \{True, False\}$
$DangerouseKitchenDevices(Oven) = True$, $DangerouseKitchenDevices(Fridge) = False$, all other values are undefined
$FrontDoorLockDevice : D \rightarrow \{True, False\}$
$FrontDoorLockDevice(FrontDoorLock) = True$, all other values are undefined
$DeviceTemperature : D \rightarrow \{x | x \textit{ is an oven temperature}\}$
$DeviceTemperature(Oven)$ is dynamically set by sensors, all other values are undefined

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

$OPA = \{KidsFriendlyContent\}$
$KidsFriendlyContent : OP \rightarrow \{True, False\}$
$KidsFriendlyContent(G_{TV}) = KidsFriendlyContent(On_{TV}) = KidsFriendlyContent(\mathit{Off}_{TV}) = True$
$KidsFriendlyContent(PG_{TV}) = KidsFriendlyContent(R_{TV}) = False$
All other values are undefined

$$
\begin{aligned}
&Authorization(s : S, op : OP, d : D, current : ES) \equiv \\
&(teenager \in FamilyRole(s) \wedge ParentInKitchen(current) = True \wedge DangerouseKitchenDevices(d) = True \wedge \\
&\qquad\qquad Device\_Temperature(d) \leq 250^{\circ}) \vee \\
&(teenager \in FamilyRole(s) \wedge DangerouseKitchenDevices(d) = False) \vee \\
&(teenager \in FamilyRole(s) \wedge FrontDoorLockDevice(d) = True \wedge FrontDoorLockToken(s) = True) \vee \\
&(teenager \in FamilyRole(s) \wedge \ (KidsFriendlyContent(op) = True \vee KidsFriendlyContent(op) = False) \vee \\
&(kid \in FamilyRole(s) \ \wedge \ (\ (day(current) \in \{Sa, S\} \wedge 12:00 \leq time(current) \leq 19:00\ ) \\
&\qquad \vee (\ day(current) \in \{M, T, W, Th, F\} \wedge 17:00 \leq time(current) \leq 19:00)\ ) \ \wedge KidsFriendlyContent(op) = True \vee \\
&(parent \in FamilyRole(s))
\end{aligned}
$$

Figure 5.10: Smart Home Use Case Configuration in $HyBAC_{AC}$

$$
\begin{aligned}
&Authorization(s:S, op:OP, d:D, current:ES) \equiv \\
&teenager \in FamilyRole(s) \wedge ParentInKitchen(current) = True \wedge \\
&\qquad DangerouseKitchenDevices(d) = True \wedge Device\_Temperature(d) \leq 250^{\circ} \vee \\
&teenager \in FamilyRole(s) \wedge DangerouseKitchenDevices(d) = False \vee \\
&teenager \in FamilyRole(s) \wedge FrontDoorLockDevice(d) = True \wedge \\
&\qquad FrontDoorLockToken(s) = True \vee \\
&teenager \in FamilyRole(s) \wedge KidsFriendlyContent(op) = True \vee \\
&teenager \in FamilyRole(s) \wedge KidsFriendlyContent(op) = False \vee \\
&kid \in FamilyRole(s) \wedge (day(current) \in \{Sa, S\} \wedge 12:00 \leq time(current) \leq 19:00) \wedge \\
&\qquad KidsFriendlyContent(op) = True \vee \\
&kid \in FamilyRole(s) \wedge (day(current) \in \{M, T, W, Th, F\} \wedge 17:00 \leq \\
&\qquad time(current) \leq 19:00) \wedge KidsFriendlyContent(op) = True \vee \\
&parent \in FamilyRole(s)
\end{aligned}
$$

Figure 5.11: The Attribute Authorization Function of The Use Case Described in Figure 5.10 in DNF Format

mentioned above we have eight conjunctive clauses, each conjunctive clause represents one access authorization rule.

To construct the authorization array we evaluate every $u_i \in U_{AC}$, $d_j \in D_{AC}$, and $op_k \in OP_{AC}$ combination against static conditions in each conjunctive clause. For every combination that satisfies every static term (condition) in a conjunctive clause we create a raw (u_i, d_j, op_k, $current$, C) for that combination in the authorization array. Where, C is the set of conditions that need to be satisfied together for a user u_i to perform an operation op_k on a device d_j. These conditions are related to the following: (a)Static or dynamic session attributes.(b)Dynamic device attributes. (c)environment attributes.

Authorizations array (AA): an authorization raw (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 session's, device's, and environment's conditions in C are satisfied. The AA of the use case in Figure 5.10 is shown in Table 5.9. For illustration purposes each different color represents the authorization fields for different user.

5.6.3 Approach

The goal is to construct $HyBAC_{RC}$ elements, assignments, derived relations, dynamic attributes sets, and attribute authorization function from $HyBAC_{AC}$ policies in such a way that the authorizations are the same as those under $HyBAC_{AC}$. The inputs are $HyBAC_{AC}$ elements U_{AC}, ES_{AC},

Table 5.9: AA for The Use Case Described in Figure 5.10

User u	Device d	Operation op	Environment state es	Conditions C
alex	TV	G	current	X
alex	TV	On_{TV}	current	X
alex	TV	Off_{TV}	current	X
alex	TV	G	current	Z
alex	TV	On_{TV}	current	Z
alex	TV	Off_{TV}	current	Z
bob	TV	G	current	$\{parent \in FamilyRole(s)\}$
bob	TV	PG	current	$\{parent \in FamilyRole(s)\}$
bob	TV	R	current	$\{parent \in FamilyRole(s)\}$
bob	TV	On_{TV}	current	$\{parent \in FamilyRole(s)\}$
bob	TV	Off_{TV}	current	$\{parent \in FamilyRole(s)\}$
bob	Oven	On_{oven}	current	$\{parent \in FamilyRole(s)\}$
bob	Oven	Off_{oven}	current	$\{parent \in FamilyRole(s)\}$
bob	Fridge	$Open_{fridge}$	current	$\{parent \in FamilyRole(s)\}$
bob	Fridge	$Close_{fridge}$	current	$\{parent \in FamilyRole(s)\}$
bob	FrontDoorLock	$Lock$	current	$\{parent \in FamilyRole(s)\}$
bob	FrontDoorLock	$Unlock$	current	$\{parent \in FamilyRole(s)\}$
anne	TV	G	current	$\{teenager \in FamilyRole(s)\}$
anne	TV	PG	current	$\{teenager \in FamilyRole(s)\}$
anne	TV	R	current	$\{teenager \in FamilyRole(s)\}$
anne	TV	On_{TV}	current	$\{teenager \in FamilyRole(s)\}$
anne	TV	OFF_{TV}	current	$\{teenager \in FamilyRole(s)\}$
anne	Oven	On_{oven}	current	Y
anne	Oven	OFF_{oven}	current	Y
anne	Fridge	$Open_{fridge}$	current	$\{teenager \in FamilyRole(s)\}$
anne	Fridge	$Close_{fridge}$	current	$\{teenager \in FamilyRole(s)\}$
anne	FrontDoorLock	$Lock$	current	L
anne	FrontDoorLock	$Unlock$	current	L

X = $\{kid \in FamilyRole(s)\ , day(current) \in \{Sa, S\}, 12:00 \leq time(current) \leq 19:00\}$.
Y = $\{teenager \in FamilyRole(s)\ , ParentInKitchen(current) = True, Device\_Temperature(d) \leq 250°\}$.
Z= $\{kid \in FamilyRole(s)\ , day(current) \in \{M, T, W, Th, F\}, 17:00 \leq time(current) \leq 19:00\}$.
L= $\{teenager \in FamilyRole(s)\ , FrontDoorLockToken(s) = True\}$.

Table 5.10: PDRA array for The Use Case Described in Figure 5.10

	DangerouseKitchenDevices = True	DangerouseKitchenDevices = False	FrontDoorLockDevice = True	FrontDoorLockDevice = False	KidsFriendlyContent = True	KidsFriendlyContent = False	RemPerm
(TV, On_{TV})	0	0	0	0	1	0	0
(TV, Off_{TV})	0	0	0	0	1	0	0
(TV, G)	0	0	0	0	1	0	0
(TV, PG)	0	0	0	0	0	1	0
(TV, R)	0	0	0	0	0	1	0
$(Oven, On_{oven})$	1	0	0	0	0	0	0
$(Oven, Off_{oven})$	1	0	0	0	0	0	0
$(Fridge, Open_{fridge})$	0	1	0	0	0	0	0
$(Fridge, Close_{fridge})$	0	1	0	0	0	0	0
$(FrontDoorLock, Lock)$	0	0	1	0	0	0	0
$(FrontDoorLock, Unlock)$	0	0	1	0	0	0	0

Table 5.11: UDRAA for The Use Case Described in Figure 5.10

	DangerouseKitchenDevices = True	DangerouseKitchenDevices = False	FrontDoorLockDevice = True	KidsFriendlyContent = True	KidsFriendlyContent = False
alex	0	0	0	$\{X, Z\}$	0
bob	$\{\{parent \in FamilyRole(s)\}\}$	$\{\{parent \in FamilyRole(s)\}\}$	$\{\{parent \in FamilyRole(s)\}\}$	$\{\{parent \in FamilyRole(s)\}\}$	$\{\{parent \in FamilyRole(s)\}\}$
anne	$\{Y\}$	$\{\{teenager \in FamilyRole(s)\}\}$	$\{L\}$	$\{\{teenager \in FamilyRole(s)\}\}$	$\{\{teenager \in FamilyRole(s)\}\}$

X = $\{kid \in FamilyRole(s)\ ,\ day(current) \in \{Sa, S\}, 12:00 \leq time(current) \leq 19:00\}$.
Y = $\{teenager \in FamilyRole(s)\ ,\ ParentInKitchen(current) = True,\ Device\_Temperature(d) \leq 250°\}$.
Z= $\{kid \in FamilyRole(s)\ ,\ day(current) \in \{M, T, W, Th, F\}, 17:00 \leq time(current) \leq 19:00\}$.
L= $\{teenager \in FamilyRole(s)\ ,\ FrontDoorLockToken(s) = True\}$.

D_{AC}, OP_{AC}, USA_{AC}, ESA_{AC}, OPA_{AC}, DA_{AC}, and the authorization array AA. The outputs are $HyBAC_{RC}$ components U_{RC}, R_{RC}, UA_{RC}, EC_{RC}, ER_{RC}, EA_{RC}, RP_{RC}, $RPRA_{RC}$, $RPEA_{RC}$, D_{RC}, OP_{RC}, P_{RC}, DR_{RC}, $PDRA_{RC}$, $RPDRA_{RC}$, $DUSA_{RC}$, DDA_{RC}, and $Authorization(s : S, op : OP, d : D)_{RC}$. The steps are similar to the steps carried in Section 4.4.2 to construct $EGRBAC$ configuration from $HABAC$ configuration as shown in the following:

Step 1: Initialization. The set of users, devices, and operations are the same in both systems, hence $U_{RC} = U_{AC}, D_{RC} = D_{AC}$, and $OP_{RC} = OP_{AC}$. 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 p_x, and add it to the set P_{RC}.

Step 2: Create the set of device roles DR_{RC}. (a) Create a device role dr for each operation attribute instance, or static device attribute instance. DR_{RC} 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 call it remaining permissions $RemPerm$ for all the permissions $p_l = (d_i, op_j)$, where $p_l \in P_{RC}$ and 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 device's or operation's attributes.

Step 3: Construct the permission device role assignment array $PDRA$. It is a many-to-many mapping of P set and DR set (constructed in Step 2). To construct $PDRA$ we first make a column for each device role $dr_j \in DR_{RC}$, and make a row for each permission $p_x \in P_{RC}$. 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 $PDRA_{RC}$ of $HyBAC_{RC}$. See Table 5.10 for the $PDRA$ array of our smart home use case shown in Figure 5.10. Finally, delete device roles that do not have any permission assigned to them, such as, $RemPerm$ device role shown in Table 5.10 .

Step 4: Construct the user device role authorization array $UDRAA$ from the authorization array AA, and $PDRA_{RC}$. $UDRAA \subseteq U \times DR$, a many to many mapping between U and DR. To construct $UDRAA$, we first make a raw 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 $(p_x, dr_j) \in PDRA_{RC}$. 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) $UDRAA[i, j] = 2^C$, where C is a set of user/session attributes, dynamic device attributes, and environment attributes conditions that need to be satisfied together for a u_i to access all the permissions assigned to dr_j. Note that these sets of conditions have to 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 5.11 shows $UDRAA$ for our use case.

Step 5: Create the set of dynamic user/session attributes $DUSA$, and the set of dynamic device attributes DDA as following:

1. Initialize the sets $DUSA_{RC}$ and DDA_{RC}
2. ($\forall$ dynamic user/session attribute $usa_i \in USA_{AC}$)$[DUSA_{RC} \leftarrow DUSA_{RC} \cup \{usa_i\}]$
3. ($\forall$ dynamic device attribute $da_i \in DA_{AC}$)$[DDA_{RC} \leftarrow DDA_{RC} \cup \{da_i\}]$

Step 6: Construct the rest of $HyBAC_{RC}$ elements $(R_{RC}, EC_{RC}, ER_{RC}, RP_{RC})$, assignments $(UA_{RC}, EA_{RC}, RPDRA_{RC})$, derived relations $(RPRA_{RC}, RPEA_{RC})$, dynamic attribute functions $DUSA_{RC}, DDA_{RC}$, and attribute authorization function $Authorization(s : S, op : OP, d : D)_{RC}$ by following our proposed $HyBAC_{RC}$ constructing algorithm introduced in Section 5.6.4. The set of users roles R_{RC} constructed here is the set of candidate users roles.

Step 7: Merge similar users roles. To do so, we run our developed role merging algorithm illustrated in Section 5.6.5.

Algorithm 3 $HyBAC_{RC}$ 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 raw i in $UDRAA$.
Require: $IsESC(c)$: Return True if c is an environment state condition.
Require: $IsDDC(c)$: Return True if c is a dynamic device attribute condition.
Require: $IsUSDC(c)$: Return True if c is a dynamic user/session attribute condition.
Require: $ToString(x)$: Return the value of x in a string format.

Initilize n = Number of users, m = Number of device roles, $R_{RC} = \{\}$, $UA_{RC} = \{\}$, $EC_{RC} = \{\}$, $ER_{RC} = \{\}$, $EA_{RC} = \{\}$, $RP_{RC} = \{\}$, $RPDRA_{RC} = \{\}$, and $Authorization(s : S, op : O, d : D) =$ "$False$"
for $j \leftarrow 1$ to m **do**
  for $i \leftarrow 1$ to n **do**
    if $UDRAA[i, j] = 1$ **then**
      $er_x = Any\_Time$, $ec_x = True$
      $EC_{EGRBAC} = EC_{EGRBAC} \cup \{ec_x\}$, $ER_{EGRBAC} = ER_{EGRBAC} \cup \{er_x\}$
      $EA_{EGRBAC} = EA_{EGRBAC} \cup \{(\{ec_x\}, er_x)\}$
      $SER = \{er_x\}$
      $r_m = ToString(ColumnDR(j))$
      $R_{EGRBAC} = R_{EGRBAC} \cup \{r_m\}$
      $RP_{EGRBAC} = RP_{EGRBAC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup \{(rp_z, ColumnDR(j))\}$
      $UA_{EGRBAC} = UA_{EGRBAC} \cup \{(RawUser(i), r_m)\}$
      $Rolepart = r_m +$ " $\in$ " $+$ "$R(s)$"
      $DRpart = ToString(ColumnDR(j)) +$ " $\in$ " $+$ "$drole((op, d)$"
      $AuthFunc \leftarrow Rolepart +$ " $\wedge$ " $+ DRpart$
      $Authorization(s : S, op : O, d : D) \leftarrow$
        $Authorization(s : S, op : O, d : D) +$ " $\vee$ " $+ AuthFunc$

    else if $UDRAA[i, j] \neq 1 \wedge UDRAA[i, j] \neq 0$ **then**
      for each $X \in UDRAA[i, j]$ **do**
        $SER = \{\}$, $AuthFunc =$ "$True$"
        for each $y \in X$ **do**
          if $IsESC(y)$ **then**
            Create ec_y, and er_y
            $EC_{RC} = EC_{RC} \cup \{ec_y\}$, $ER_{RC} = ER_{RC} \cup \{er_y\}$
            $EA_{RC} = EA_{RC} \cup \{(\{ec_y\}, er_y)\}$
            $SER = SER \cup \{er_y\}$
          else if $IsUSDC(y)$ **then**
            $AuthFunc \leftarrow AuthFunc +$ " $\wedge$ " $+ ToString(y)$
          else if $IsDDC(y)$ **then**
            $AuthFunc \leftarrow AuthFunc +$ " $\wedge$ " $+ ToString(y)$
          end if
        end for

35: **if** $SER == \{\}$ **then**
36: $er_x = Any\_Time, ec_x = True$
37: $EC_{RC} = EC_{RC} \cup \{ec_x\}, ER_{RC} = ER_{RC} \cup \{er_x\}$
38: $EA_{RC} = EA_{RC} \cup \{(\{ec_x\}, er_x)\}$
39: $SER = \{er_x\}$
40: **end if**
41: $r_m = ToString(ColumnDR(j)) +" \wedge "+ ToString(X)$
42: $R_{RC} = R_{RC} \cup \{r_m\}$
43: $RP_{RC} = RP_{RC} \cup \{rp_z\}$, where $rp_z.r = r_m, rp_z.ER = SER$
44: $RPDRA_{RC} = RPDRA_{RC} \cup (rp_z, ColumnDR(j))$
45: $UA_{RC} = UA_{RC} \cup \{(RawUser(i), r_m)\}$
46: $Rolepart = r_m +" \in " +"R(s)"$
47: $DRpart = ToString(ColumnDR(j)) +" \in " +"drole((op, d)"$
48: $AuthFunc \leftarrow Rolepart +" \wedge " + AuthFunc +" \wedge " + DRpart$
49: $Authorization(s : S, op : O, d : D) \leftarrow$
50: $Authorization(s : S, op : O, d : D) +" \vee " + AuthFunc$
51: **end for**
52: **end if**
53: **end for**
54: **end for**

5.6.4 $HyBAC_{RC}$ Roles and Authorization function Constructing Algorithm

The input is $UDRAA$. The goal is to Construct $HyBAC_{RC}$ elements $(R_{RC}, EC_{RC}, ER_{RC}, RP_{RC})$, assignments $(EA_{RC}, RPDRA_{RC})$, derived relations $(RPRA_{RC}, RPEA_{RC})$, and attribute authorization function $Authorization(s : S, op : OP, d : D)_{RC}$ from $UDRAA$. See Algorithm 3 for the full algorithm. The steps are shown as following:

Step 1: Initialize the attribute authorization function and the following $HyBAC_{RC}$ sets $R_{RC} = \{\}, UA_{RC} = \{\}, EC_{RC} = \{\}, ER_{RC} = \{\}, EA_{RC} = \{\}, RP_{RC} = \{\}, RPDRA_{RC} = \{\}$. Moreover, initialize the following constants $m =$ Number of device roles, $n =$ Number of users.

Step 2: Loop through the columns of $UDRAA$, Table 5.11 for the use case introduced in Figure 5.10. Each column is corresponding to users access rights to a specific device role. Inside each column loop through the fields of different rows. Here, if the field is equal to zero, according to the way $UDRAA$ was constructed this means the user corresponding to this field has no access right to this field device role, hence, the algorithm go to the next field. Otherwise, we have two cases:

A. $UDRAA[i, j] = 1$**,** according to the way $UDRAA$ was constructed this means the user corre-

sponding to this raw u_i can access the device role of this column dr_j unconditionally. In this case, the algorithm does the following:

a Creates an environment role $er_x = Any\_Time$ and add it to the set ER_{RC}, an environment condition $ec_x = True$ and add it to the set EC_{RC}. Add $(\{ec_x\}, er_x)$ to the set EA_{RC}, 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 $SER = \{er_x\}$.

b Creates 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_{RC}.

c Defines a role pair rp_z, where $rp_z.r = r_m$ and $rp_z.ER = SER$. Add rp_z to the set RP_{RC}.

d Assigns the role pair rp_z to the device role corresponding to this column by adding the pair $(rp_z, ColumnDR(j))$ to the set $RPDRA_{RC}$.

e Assigns the role r_m to the user corresponding to this raw by adding the pair$(RawUser(i), r_m)$ to the set UA_{RC}.

f Creates an authorization policy that permits users assigned to r_m to access permissions assigned to dr_j unconditionally. Finally add this policy to the authorization function.

B. $UDRAA[i,j] \neq 1 \wedge UDRAA[i,j] \neq 0$**,** which means that the user u_i can access the device role dr_j under specific set of user, devices, and environment conditions defined by $UDRAA[i,j]$. Here, $UDRAA[i,j]$ is a set of sets of conditions, each set of conditions define a group of session, devices, and environment condition that need to be satisfied together in order for the user u_i 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 $y \in X$, and check for the following options:

   a If y is an environment state attribute condition, the algorithm creates a corresponding environment condition ec_y and adds it to the set EC_{RC}, environment role er_y and adds it

to the set ER_{RC}. Adds $(\{ec_x\}, er_x)$ to the set EA_{RC}. Moreover, the algorithm adds er_y to the set of environment roles SER.

b If y is a dynamic session condition, the algorithm updates the authorization function $AuthFunc$ accordingly by logically conjunct this session condition to the authorization function.

c If y is a dynamic device attribute condition, the algorithm updates the authorization function $AuthFunc$ accordingly by logically conjunct this dynamic device attribute condition to the authorization function.

2. After looping through each condition in X, if the set SER is empty, that means this set of conditions does not contain an environment state condition. In other words, the user of this raw 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_{RC}, an environment condition $ec_x = True$ and add it to the set EC_{RC}. Add $(\{ec_x\}, er_x)$ to the set EA_{RC}, 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_{RC}.

4. Defines a role pair rp_z, where $rp_z.r = r_m$ and $rp_z.ER = SER$. Add rp_z to the set RP_{RC}.

5. Assigns the role pair rp_z to the device role corresponding to this column by adding the pair $(rp_z, ColumnDR(j))$ to the set $RPDRA_{RC}$.

6. Assigns the role r_m to the user corresponding to this raw by adding the pair$(RawUser(i), r_m)$ to the set UA_{RC}.

7. Finally, the algorithm prepare the authorization function corresponding to this field of $UDRAA$ and then add it to the final authorization function.

5.6.5 Users Roles Merging Algorithm

Algorithm 4 Users Roles Merging 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. .

for each $r_i, r_j \in R_{EGRBAC}$ **do**
  if $U(r_i) = U(r_j) \wedge r_i \neq r_j$ **then**
    for each $rp_k \in RP(r_i)$ **do**
      $rp_k.r = r_j$
    end for

    $R_{EGRBAC} = R_{EGRBAC} \setminus r_i$

    ▷ Delete all UA pairs related to r_i
    for each $(u_l, r_i) \in UA_{EGRBAC}$ **do**
      $UA = UA \setminus (u_l, r_i)$
    end for

    ▷ Replace all $RPDRA$ pairs related to r_i
    for each $((r_i, ER_x), dr_y) \in RPDRA_{EGRBAC}$ **do**
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \setminus ((r_i, ER_x), dr_y)$
      $RPDRA_{EGRBAC} = RPDRA_{EGRBAC} \cup \{((r_j, ER_x), dr_y)\}$
    end for
    Replace every r_i in the authorization function with r_j.
  end if
end for

The main purpose of this algorithm is to merge roles that have similar users assignments. For each two roles r_i, r_j which are assigned to the same set of users, the algorithm does 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_{RC}.

3. For every $(u_l, r_i) \in UA_{RC}$, remove the pair (u_l, r_i) from the set UA_{RC}. See Algorithm 4 for the complete algorithm.

4. For every $((r_i, ER_x), dr_y) \in RPDRA_{RC}$, remove $(r_i, ER_x), dr_y)$ from the set $RPDRA_{RC}$, and instead add the pair $(r_j, ER_x), dr_y)$ to the set $RPDRA_{RC}$.

$r_1 \equiv DangerouseKitchenDevices = True \wedge parent \in FamilyRole(s)$,
$r_2 \equiv DangerouseKitchenDevices = True \wedge \{teenager \in FamilyRole(s)$,
$\quad ParentInKitchen(current) = True \wedge DeviceTemperature(d) \leq 250°\}$,
$r_3 \equiv DangerouseKitchenDevices = False \wedge parent \in FamilyRole(s)$,
$r_4 \equiv DangerouseKitchenDevices = False \wedge teenager \in FamilyRole(s)$,
$r_5 \equiv FrontDoorLockDevice = True \wedge parent \in FamilyRole(s)$,
$r_6 \equiv FrontDoorLockDevice = True \wedge \{teenager \in FamilyRole(s)$,
$\quad FrontDoorLockToken(s) = True\}$,
$r_7 \equiv KidsFriendly = True \wedge \{kid \in FamilyRole(s)$,
$\quad day(current) \in \{Sa, S\}, 12:00 \leq time(current) \leq 19:00\}$,
$r_8 \equiv KidsFriendly = True \wedge \{kid \in FamilyRole(s)$,
$\quad day(current) \in \{M, T, W, Th, F\}, 17:00 \leq time(current) \leq 19:00\}$,
$r_9 \equiv KidsFriendly = True \wedge parent \in FamilyRole(s)$,
$r_{10} \equiv KidsFriendly = True \wedge teenager \in FamilyRole(s)$,
$r_{11} \equiv KidsFriendly = False \wedge parent \in FamilyRole(s)$,
$r_{12} \equiv KidsFriendly = False \wedge teenager \in FamilyRole(s)$,

Figure 5.12: Initial Set of Roles Before Running the Role Merging Algorithm

$Authorization(s : S, op : OP, d : D) \equiv$
$(r_1 \in R(s) \wedge DangerouseKitchenDevices = True \in drole((op, d)) \vee$
$(r_2 \in R(s) \wedge DeviceTemperature(d) \leq 250° \wedge; DangerouseKitchenDevices = True \in drole((op, d))) \vee$
$(r_3 \in R(s) \wedge DangerouseKitchenDevices = False \in drole((op, d)) \vee$
$(r_4 \in R(s) \wedge DangerouseKitchenDevices = False \in drole((op, d)) \vee$
$(r_5 \in R(s) \wedge FrontDoorLockDevice = True \in drole((op, d)) \vee$
$(r_6 \in R(s) \wedge FrontDoorLockToken(s) = True \wedge; FrontDoorLockDevice = True \in drole((op, d)) \vee$
$(r_7 \in R(s) \wedge KidsFriendlyContent = True \in drole((op, d))) \vee$
$(r_8 \in R(s) \wedge KidsFriendlyContent = True \in drole((op, d))) \vee$
$(r_9 \in R(s) \wedge KidsFriendlyContent = True \in drole((op, d)) \vee$
$(r_{10} \in R(s) \wedge KidsFriendlyContent = True \in drole((op, d)) \vee$
$(r_{11} \in R(s) \wedge KidsFriendlyContent = False \in drole((op, d)) \vee$
$(r_{12} \in R(s) \wedge KidsFriendlyContent = False \in drole((op, d))$

Figure 5.13: Initial Authorization function Before Running the Role Merging Algorithm

5. Finally, update the authorization function accordingly, by replacing any r_i with r_j.

After applying the first six steps of the approach of constructing $HyBAC_{RC}$ from $HyBAC_{AC}$ introduced in Section 5.6.3 on our smart home use case introduced in Figure 5.10, we will end up having a set of twelve roles as illustrated in Figure 5.12. These roles will be assigned to different users as following:

$$UA = \{(alex, r_7), (alex, r_8), (bob, r_1), (bob, r_3), (bob, r_5), (bob, r_9), (bob, r_{11}),$$
$$(anne, r_2), (anne, r_4), (anne, r_6), (anne, r_{10}), (anne, r_{12})\}.$$

Moreover, we will end up having the dynamic attribute authorization function shown in Figure 5.13. After running the users roles merging algorithm, the constructed twelve roles will be merged

$Authorization(s:S, op:OP, d:D) \equiv$
$(r_a \in R(s) \wedge DangerouseKitchenDevices = True \in drole((op,d)) \vee$
$(r_b \in R(s) \wedge DeviceTemperature(d) \leq 250° \wedge; DangerouseKitchenDevices = True \in drole((op,d))) \vee$
$(r_a \in R(s) \wedge DangerouseKitchenDevices = False \in drole((op,d)) \vee$
$(r_b \in R(s) \wedge DangerouseKitchenDevices = False \in drole((op,d)) \vee$
$(r_a \in R(s) \wedge FrontDoorLockDevice = True \in drole((op,d)) \vee$
$(r_b \in R(s) \wedge FrontDoorLockToken(s) = True \wedge; FrontDoorLockDevice = True \in drole((op,d)) \vee$
$(r_c \in R(s) \wedge KidsFriendlyContent = True \in drole((op,d))) \vee$
$(r_c \in R(s) \wedge KidsFriendlyContent = True \in drole((op,d))) \vee$
$(r_a \in R(s) \wedge KidsFriendlyContent = True \in drole((op,d)) \vee$
$(r_b \in R(s) \wedge KidsFriendlyContent = True \in drole((op,d)) \vee$
$(r_a \in R(s) \wedge KidsFriendlyContent = False \in drole((op,d)) \vee$
$(r_b \in R(s) \wedge KidsFriendlyContent = False \in drole((op,d))$

Figure 5.14: Authorization function After Running the Role Merging Algorithm

into three roles only, and the user role assignment set will end up having three pairs as shown in the following:

$$R = \{ r_a \equiv r_1, r_3, r_5, r_9, r_{11}, r_b \equiv r_2, r_4, r_6, r_{10}, r_{12}, r_c \equiv r_7, r_8\}.$$
$$UA = \{(alex, r_c), (bob, r_a), (anne, r_b)\}.$$

Figure 5.14 shows the final authorization function after running the role merging algorithm.

5.6.6 The output of $HyBAC_{RC}$ Constructing Approach on $HyBAC_{AC}$ Use Case

The output of $HyBAC_{RC}$ role constructing algorithm for the Use case in Table 5.10 is shown in Table 5.12.

5.7 A Comprehensive Comparison

Considering the diverse structure, requirements, and specifications of an organization, and taking into account that access control policies and models are available in diverse forms, it is required to select and implement an appropriate access control model consistent with the security requirements of the related organization in order to achieve the best results and minimum access risks and threats. In this section, we compare between four access control models specifically designed to meet smart home IoT challenges. We compare the two hybrid models developed in this paper,

$HyBAC_{RC}$, and $HyBAC_{AC}$ with $EGRBAC$ and $HABAC$. We evaluate these models against an access control criteria adapted from [58]. These criteria are classified into two types: (a) Basic and main criteria. (b) Quality criteria.

5.7.1 Basic and main criteria

This type of criteria consists of six elements. Here we discuss each criterion.

Constraints Constraints are invariants that must be maintained. We have three types of constraints, static separation of duty, dynamic separation of duty, and permission-role constraints as explained in Section 5.2, and Section 5.3. From table 5.13, we can notice that all four models support the three types of constraints except for $HABAC$. Where it does not support permission-role constraints [20].

Attributed based specifications We have two types of attributes as explained in Section 5.1, static and dynamic. The four models support environment attributes. Moreover, they all support static users, devices, and operations attributes (conditions). On the other hand, dynamic attributes are not supported by $EGRBAC$.

Least privilege principle This principle means that a subject of a system should only be permitted to have access to the least privileges that are required for performing the user's duties. All four models support this principle. They all include the component session, and a user belonging to several roles (in $EGRBAC$, and $HyBAC_{RC}$), or has different attributes corresponding to his roles in the house (in $HABAC$, and $HyBAC_{AC}$) can invoke any subset of them that enables tasks to be accomplished in a session. Thus, a user who is a member of a powerful role can normally keep this role deactivated and explicitly activate it when needed.

Authentication All four models support positive (close) authentication. Closed policy permits an access when there is a positive authorization for such access, and denied it otherwise.

Access administration Here we compare the four models based on three administrative tasks. User provisioning, policy provisioning, and configuration effort. In general, user provisioning is

easier in RBAC based models (including $EGRBAC$ and $HyBAC_{RC}$) than in ABAC based models (including $HABAC$, and $HyBAC_{AC}$). In RBAC models user provisioning simply requires the administrator (the homeowner) assigns users to roles. On the other hand, in ABAC based models, the administrator needs to configure different attribute values for newly provisioned users, and devices. On the contrary, in ABAC based model policy provisioning only require the addition of those policies to the authorization function as in $HABAC$ or to the authorization set as in $HyBAC_{AC}$. Differently, in RBAC based model this requires configuring a series of assignments as in $EGRBAC$, and configuring a series of assignments and adding new authorization rules as in $HyBAC_{RC}$. Regarding configuration effort, from Table 5.13, we can notice that $HyBAC_{RC}$ requires more configuration effort than the other three models. On the other hand, $EGRBAC$ (the pure RBAC model) is the one with the least configuration efforts.

Access review In RBAC based model (including $EGRBAC$ and $HyBAC_{RC}$), to determine the maximum permission available for a user we just need to look into his roles while this could be more complicated in ABAC based models (such as $HABAC$ and $HyBAC_{AC}$) [70].

Administrative policies To determine how administrative privileges are organized in any model, an access control administration model is required. However, in the four models, it is assumed that the house owner is the one who is responsible for granting or revoking permissions. Hence, we can say that they all support centralized administrative policies.

5.7.2 Quality criteria

Here we have three important criteria as explained in the following.

Expressiveness, and Meaningfulness We believe that in order for an AC model to be expressive, it has to maintain at least the following three characteristics. First, it has to be formally defined so that there is a precise and rigorous specification of the intended behavior. Second, it has to be sufficiently meaningful and expressive to support different types of constraints. Finally, The model should capture different types of static and dynamic attributes. All four models that we are comparing are formally defined. Furthermore, they all support different constraints except for $HABAC$,

since it does not support permission-role constraints. Finally, as explained in the Attributed based specifications criterion in Section 5.7.1. They all capture different types of attributes except for $EGRBAC$, where it doesn't capture the user's and device's dynamic attributes.

Flexibility To consider a specific AC model as a flexible model several factors need to be evaluated, here we identify three of them. First, the model should be flexible enough to meet smart home IoT requirements. Second, the model should support delegation which means the capability of a subject to delegate his privileges to any other user partially or totally. Moreover, the flexibility of provisioning new users or policies.

According to the criteria proposed in Section **??** 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 otherwise be interpreted for the smart home domain such as by 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 there is a precise and rigorous specification of the intended behavior. As discussed in [19] $EGRBAC$ meet this criteria. On the other hand, $HABAC$ model was not implemented or tested, hence it doesn't fulfill the entire criteria. Moreover, $HyBAC_{RC}$, and $HyBAC_{AC}$ are both dynamic, fine grained, suitable for the constrained home environment, designed specifically for smart home IoT, illustrated with a use case demonstration, has proof of concept implementation, and they are both formally defined. Hence, they meet the criteria proposed in [19]. From the above, we can claim that except for HABAC, these models are flexible enough to meet smart home IoT requirements. HABAC needs to be implemented and tested to ensure its applicability using commercially available technology.

Regarding delegation support, this can't be fully determined without an access control administration model. But generally speaking, it has been shown before that RBAC and ABAC based models are capable of performing delegation. All four models are capable of provisioning new users and policies. However, as discussed earlier, while it is easier to provision new users in RBAC based models (including $EGRBAC$ and $HyBAC_{RC}$) than in ABAC based models (in-

cluding $HABAC$ and $HyBAC_{AC}$), it is more complicated to provision new policies in RBAC based models than in ABAC based models.

Efficiency level and scalability An access control model is required to answer two main questions on efficiency level and scalability. If the model cannot be expanded easily, this program will be questionable in the real world. Moreover, the development of the model should not affect its efficiency negatively. To answer these questions accurately a more detailed study needs to be performed. However, generally speaking in smart home IoT, we are dealing with a relatively small number of users. Moreover, ABAC based models and RBAC based models have proved their scalability since they have been widely adopted on different organizations with huge sizes.

5.8 Analysis and Limitations

In this chapter, we proposed $HyBAC_{RC}$. It is a user to device access control model. It is a role-centric hybrid model that combines the basic concepts of EGRBAC (an RBAC model) and HABAC (an ABAC model) systems. It encapsulates relatively static attributes of access decisions in user roles and device roles. It utilizes the concept of user's and device's dynamic attributes to capture rapidly changing attributes to further constrain the permissions available for each user. The users' roles and devices' role sets, therefore, determine the maximum set of available permissions, supporting the principle of least privilege and allowing easy review of user permissions. Moreover, $HyBAC_{RC}$ can capture the environment contextual information through the component environment roles. $HyBAC_{RC}$ retains advantages of EGRBAC (such as ease of user provisioning, least privilege principle, and the ability to quickly determine and control the maximum permissions available to each user) while captures different authorizations for every possible user, environment, operation, and device dynamic conditions without involving the risk of role explosion.

Moreover, we introduced $HyBAC_{AC}$. It is an attribute-centric user to device access control model that is based on the HABAC model proposed in Chapter 4. It captures different users', environment's, operations', and devices' static and dynamic characteristics. Therefore, it is a dynamic model. It is a fine grained model; since it is capable of giving users access to some operations

within a single device without the need to give them access to the entire device. Unlike HABAC, $HyBAC_{AC}$ supports the permission-role constraints by introducing the notion of roles (aka anti-roles). However, it enforces this type of constraint during execution time. On the other hand, $EGRBAC$, and $HyBAC_{RC}$ enforces permission-role constraints during configuration time.

Our approach of constructing $HyBAC_{AC}$ from $HyBAC_{RC}$ is a simple, straightforward approach that is capable of translating $HyBAC_{RC}$ configuration into an $HyBAC_{AC}$ policy configuration.

In addition to static users/sessions, devices, and operations attributes, our $HyBAC_{RC}$ constructing approach is capable of handling $HyBAC_{RC}$ policies that contain environment attributes, and dynamic user/session, and devices attributes. On the other hand, our $HyBAC_{RC}$ constructing approach didn't consider the following: (1)Policies that compare two different types of attributes. (2)$HyBAC_{AC}$ configurations that involve constraints. However, this may be a possible future direction.

Table 5.12: The output of $HyBAC_{RC}$ Constructing Approach on The Use Case Described in Figure 5.10

(a) $U_{RC} = U_{AC}, D_{RC} = D_{AC}, OP_{RC} = OP_{AC}$, $P_{RC} = \{(TV, G)$, (TV, PG), (TV, R), (TV, On_{TV}), (TV, Off_{TV}), $(Oven, On_{oven})$, $(Oven, Off_{oven})$, $(Fridge, Open_{fridge})$, $(Fridge, Close_{fridge})$, $(FrontDoorLock, Lock)$, $(FrontDoorLock, Unlock)\}$

(b) $DR = \{DangerouseKitchenDevices = True$, $DangerouseKitchenDevices = False$, $FrontDoorLockDevice = True$, $FrontDoorLockDevice = False$, $KidsFriendlyContent = True$, $KidsFriendlyContent = False$, $RemPerm\}$.

(c) $PDRA = \{((TV, G), KidsFriendly = True)$, $((TV, On_{TV}), KidsFriendly = True)$, $((TV, Off_{TV}), KidsFriendly = True)$, $((TV, PG), KidsFriendly = False)$, $((TV, R), KidsFriendly = False)$, $((Oven, On_{oven}), DangerouseKitchenDevices = True)$, $((Oven, Off_{oven}), DangerouseKitchenDevices = True)$, $((Fridge, Open_{fridge}), DangerouseKitchenDevices = False)$, $((Fridge, Close_{fridge}), DangerouseKitchenDevices = False)$, $((FrontDoorLock, Lock), FrontDoorLockDevice = True)$, $((FrontDoorLock, Unlock), FrontDoorLockDevice = True)\}$.

(d) $EC = \{True$, $ec_1 \equiv ParentInKitchen(current) = True$, $ec_2 \equiv day(current) \in \{Sa, S\}$, $ec_3 \equiv 12:00 \leq time(current) \leq 19:00$, $ec_4 \equiv day(current) \in \{M, T, W, Th, F\}$, $ec_5 \equiv 17:00 \leq time(current) \leq 19:00\}$.

(e) $ER = \{Any\_Time$, $er_1 \equiv ParentInKitchen(current) = True$, $er_2 \equiv day(current) \in \{Sa, S\} \equiv Weekend$, $er_3 \equiv 12:00 \leq time(current) \leq 19:00 \equiv Afternoon\ and\ Evening$, $er_4 \equiv day(current) \in \{M, T, W, Th, F\} \equiv Weekdays$, $er_5 \equiv 17:00 \leq time(current) \leq 19:00 \equiv Evening\}$.

(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) $UA = \{(bob, r_a), (anne, r_b), (alex, r_c)\}$.

(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) $RPDRA = \{((r_a, Any_Time)$, $DangerouseKitchenDevices = True)$, $((r_a, Any\_Time)$, $DangerouseKitchenDevices = False)$, $((r_a, Any\_Time)$, $FrontDoorLockDevice = True)$, $((r_a, Any\_Time)$, $KidsFriendly = True)$, $((r_a, Any\_Time)$, $KidsFriendly = False)$, $((r_b, \{er_1\})$, $DangerouseKitchenDevices = True)$, $((r_b, Any\_Time)$, $DangerouseKitchenDevices = False)$, $((r_b, Any\_Time)$, $FrontDoorLockDevice = True)$, $((r_b, Any\_Time)$, $KidsFriendly = True)$, $((r_b, Any\_Time)$, $KidsFriendly = False)$, $((r_c, \{er_2, er_3\})$, $KidsFriendly = True)$, $((r_c, \{er_4, er_5\})$, $KidsFriendly = True)\}$.

Table 5.13: Evaluating Smart Home IoT Access Control Models Against Basic Criteria

Criteria	$EGRBAC$	$HABAC$	$HyBAC_{RC}$	$HyBAC_{AC}$
1. Constraints				
a. Static separation of duty	Yes	Yes	yes	Yes
b. Dynamic separation of duty	Yes	yes	yes	yes
c. Least privilege	Yes	yes	yes	yes
d. P-R constraints	Yes	No	yes	yes
2. Attributed based specifications				
a. Static	Yes	Yes	yes	Yes
b. Dynamic	No	yes	yes	yes
3. Least privilege principle	Yes	yes	yes	yes
4. Authentication	Positive(Close)	Positive(Close)	Positive(Close)	Positive(Close)
5. Access administration				
a. User provisioning	Easy	Complicated	Easy	Complicated
b. Policy provisioning	Complicated	Easy	Complicated	Easy
c. Configuration effort	1- Define and set initial users, devices, and operations static characteristics (user roles, and device roles) 2- Define environment conditions, environment roles, and environment activations 3- Setting up initial role structure and assignments	1- Define and set initial users, devices, and operations static characteristics (user roles, and device roles) 2- Define and set initial users, and devices dynamic characteristics (Dynamic attributes) 3- Define environment states, and environment state attributes 4- Specify access policies	1- Define and set initial users, devices, and operations static characteristics (attributes) 2- Define and set initial users, and devices dynamic characteristics (Dynamic attributes) 3- Define environment conditions, environment roles, and environment activations 4- Setting up initial role structure and assignments 5- Specify access policies	1- Define and set initial users, devices, and operations static characteristics (attributes) 2- Define and set initial users, and devices dynamic characteristics (Dynamic attributes) 3- Define environment states, and environment state attributes 4- Specify access policies
6. Access review	Easy	Complicated	Easy	Complicated
7. Administrative policies	Centralized	Centralized	Centralized	Centralized

CHAPTER 6: CONCLUSION AND FUTURE WORK

This chapter concludes our dissertation. It summarizes the contributions of this dissertation and presents future research directions that can be further investigated.

6.1 summary

This dissertation makes fundamental contributions towards addressing the lack of access control models suitable for smart home IoT. It develops a family (or series) of models ranging from relatively simple to incorporating increasingly sophisticated and comprehensive features. Our intuitive insight was that a hybrid model which combines both ABAC and RBAC components will better capture smart home IoT access control requirements. In order to further investigate this intuition our approach was to develop pure RBAC and pure ABAC based models explicitly defined to meet smart home challenges, and compare their benefits and drawbacks. This comparison provided insights to guide us in designing adequate hybrid models to meet smart home IoT challenges.

Initially, we started by analyzing IoT access control models proposed in the literature, and smart home IoT access control challenges and formulated criteria that need to be maintained in future proposed access control models.

Second, we introduced the extended generalized role based access control ($EGRBAC$) model for user to device interaction in smart home IoT. It is a dynamic, fine-grained model that grants access based on the specific permission required rather than at device granularity. We demonstrated our model with a use case scenario and a proof-of-concept implementation in AWS. We also conducted a performance test to depict how our system responds in different scenarios with different loads. The results show that our model is functional and applicable in practice.

Third, we proposed $HABAC$. It is an attribute-based access control model for smart home IoT. It is provided with a use case scenario demonstration. Moreover, we compared the theoretical expressive power of $HABAC$ to $EGRBAC$ by providing approaches for converting an $HABAC$ specification to $EGRBAC$ and vice versa. We found that while $EGRBAC$ is capable of handling

environment attributes, and relatively static user, and device attributes, it is incapable of handling relatively dynamic users and devices attributes. On the other hand, unlike $EGRBAC$, in $HABAC$ it is hard to prevent future authorization of specific users to access specific operations on specific devices. Hence, reinforcing our argument for a hybrid approach.

As a result, our fourth contribution was presenting two hybrid models for smart home IoT access control $HyBAC_{RC}$, and $HyBAC_{AC}$. While $HyBAC_{RC}$ is a role-centric combined (RBAC and ABAC) access control model, $HyBAC_{AC}$ is an attribute-centric combined (RBAC and ABAC) access control model. These models were formally defined, illustrated through use case scenarios, and implemented and tested in Amazon Web Services (AWS). Moreover, we compared the theoretical expressive power of $HyBAC_{AC}$, and $HyBAC_{RC}$ models by providing algorithms for converting an $HyBAC_{AC}$ specification to $HyBAC_{RC}$ and vice versa. We discussed the insights for practical deployment of these models. Finally, we provided a comprehensive theoretical comparison between the four smart home IoT access control models introduced in this dissertation.

6.2 Future Work

There are several potential directions that can be studied and explored as extensions to this research. This dissertation focuses on user to device interaction in smart home IoT. However, analogous to the way humans use the Internet, devices will be the main "users" on the Internet of Things ecosystem. These devices will operate with different networking standards which will increase the risk exposure for their communication. Therefore, device-to-device (D2D) communication is expected to be an intrinsic part of IoT. Devices will communicate with each other autonomously with or without any centralized control and collaborate to gather, share and forward information. Hence, further research on device-to-device interaction access and communication control in smart home IoT is a viable direction.

Moreover, communicating smart devices are usually constrained in resources, and they will probably operate with different networking standards. Such interaction poses heterogeneity problems. Therefore, in many cases it may be wiser to transfer the communication to the virtual objects

layer. This will provide a uniform abstract interface for the physical objects to communicate with each other, and with the upper layers. This abstraction will introduce many types of interactions that need to be secured, such as Objects-to-Virtual objects, Virtual objects-to-Virtual objects, and Virtual objects-to-Applications.

On the other hand, many researchers who studied different IoT platforms for applications to devices interactions concluded that applications usually have over privileged access to smart devices. They also figured that apps might pose a higher risk to users than needed [44, 95]. Hence, rigorous access control models for Users-to-Applications, Applications-to-Virtual objects, and Applications-to-Applications need to be developed.

Finally, the integration between IoT and cloud services will enlarge the attack surface for smart things. The cloud services layer allows IoT to leverage its practically unlimited storage, computation, and analysis capabilities. It provides the flexibility and scalability needed for IoT. This will introduce new interactions that need to be secured, for example, Cloud-to-Virtual objects, Applications-to-Cloud, Users-to-Cloud, and Cloud-to-Cloud.

BIBLIOGRAPHY

[1] OpenStack. https://www.openstack.org/.

[2] The Transport Layer Security (TLS) Protocol. https://tools.ietf.org/html/rfc5246.

[3] Amazon Web Services (AWS) - Cloud Computing Services. https://aws.amazon.com.

[4] AWS-IoT. https://aws.amazon.com/iot/.

[5] AWS IoT Device SDK for Python. https://docs.aws.amazon.com/greengrass/latest/developerguide/IoT-SDK.html.

[6] AWS IoT Greengrass. https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html.

[7] AWS lambda function. https://aws.amazon.com/lambda/.

[8] Internet of Battlefield Things. https://www.arl.army.mil/business/collaborative-alliances/current-cras/iobt-cra/.

[9] Internet of things. https://en.wikipedia.org/wiki/Internet_of_things.

[10] Microsoft Azure. https://azure.microsoft.com.

[11] MQTT.fx - A JavaFX based MQTT Client. https://softblade.de/en/welcome/.

[12] Muhammad Umar Aftab, Yasir Munir, Ariyo Oluwasanmi, Zhiguang Qin, Muhammad Haris Aziz, Ngo Tung Son, et al. A hybrid access control model with dynamic COI for secure localization of satellite and iot-based vehicles. *IEEE Access*, 2020.

[13] Tahmina Ahmed, Ravi Sandhu, and Jaehong Park. Classifying and comparing attribute-based and relationship-based access control. In *CODASPY '17*. ACM, 2017.

[14] Shadab Alam, Shams Tabrez Siddiqui, Ausaf Ahmad, Riaz Ahmad, and Mohammed Shuaib. Internet of things (IoT) enabling technologies, requirements, and security challenges. In *Advances in data and information sciences*. Springer, 2020.

[15] Frances K Aldrich. Smart homes: past, present and future. In *Inside the smart home*. Springer, 2003.

[16] Gauhar Ali, Naveed Ahmad, Yue Cao, Muhammad Asif, Haitham Cruickshank, and Qazi Ejaz Ali. Blockchain based permission delegation and access control in internet of things (BACI). *Computers & Security*, 2019.

[17] Mousa Alramadhan and Kewei Sha. An overview of access control mechanisms for internet of things. In *ICCCN*. IEEE, 2017.

[18] Mohammed Alshahrani and Issa Traore. Secure mutual authentication and automated access control for iot smart home using cumulative keyed-hash chain. *Journal of information security and applications*, 2019.

[19] Safwa Ameer, James Benson, and Ravi Sandhu. The EGRBAC model for smart home IoT. In *2020 IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI)*. IEEE, 2020.

[20] Safwa Ameer and Ravi Sandhu. The HABAC model for smart home IoT and comparison to EGRBAC. In *ACM Workshop on Secure and Trustworthy Cyber-Physical Systems (SAT-CPS)*, 2021.

[21] Orlando Arias, Jacob Wurm, Khoa Hoang, and Yier Jin. Privacy and security in internet of things and wearable devices. *TMSCS*, 2015.

[22] Saurabh Bagchi, Tarek F Abdelzaher, Ramesh Govindan, Prashant Shenoy, Akanksha Atrey, Pradipta Ghosh, and Ran Xu. New frontiers in IoT: Networking, systems, reliability, and security challenges. *IEEE Internet of Things Journal*, 2020.

[23] Syafril Bandara, Takeshi Yashiro, Noboru Koshizuka, and Ken Sakamura. Access control framework for api-enabled devices in smart buildings. In *APCC*. IEEE, 2016.

[24] Ezedin Barka and Ravi Sandhu. Framework for role-based delegation models. In *ACSAC*. IEEE, 2000.

[25] Ezedine Barka, Sujith Samuel Mathew, and Yacine Atif. Securing the web of things with role-based access control. In *C2SI*. Springer, 2015.

[26] Bruhadeshwar Bezawada, Kyle Haefner, and Indrakshi Ray. Securing home IoT environments with attribute-based access control. In *ABAC'18*. ACM, 2018.

[27] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. Access control model for AWS internet of things. In *International Conference on Network and System Security*, 2017.

[28] Smriti Bhatt and Ravi Sandhu. 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*, 2020.

[29] K. Z. Bijon, R. Krishnan, and R. Sandhu. Towards an attribute based constraints specification language. In *SOCIALCOM '13*, Sep. 2013.

[30] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. Label-based access control: An abac model with enumerated authorization policy. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 1–12, 2016.

[31] Enrico Carniani, Davide D'Arenzo, Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control on cloud systems. *Future Gen. Comp. Sys.*, 2016.

[32] Poornima M Chanal and Mahabaleshwar S Kakkasageri. Security and privacy in IoT: A survey. *Wireless Personal Communications*.

[33] Arnab Chatterjee, Yash Pitroda, and Manojkumar Parmar. Dynamic role-based access control for decentralized applications. In *International Conference on Blockchain*. Springer, 2020.

[34] Yuan Cheng, Jaehong Park, and Ravi Sandhu. Relationship-based access control for online social networks: Beyond user-to-user relationships. In *SocialCom*. IEEE, 2012.

[35] Michael J Covington, Wende Long, Srividhya Srinivasan, Anind K Dev, Mustaque Ahamad, and Gregory D Abowd. Securing context-aware applications using environment roles. In *SACMAT '01*. ACM, 2001.

[36] Michael J Covington, Matthew James Moyer, and Mustaque Ahamad. Generalized role-based access control for securing future applications. Technical report, Georgia Tech, 2000.

[37] Ang Cui and Salvatore J Stolfo. A quantitative analysis of the insecurity of embedded network devices: results of a wide-area scan. In *ACSAC '10*. ACM, 2010.

[38] Luciana Moreira Sá De Souza, Patrik Spiess, Dominique Guinard, Moritz Köhler, Stamatis Karnouskos, and Domnic Savio. Socrades: A web service based shop floor integration infrastructure. In *The IoT*. Springer, 2008.

[39] Tamara Denning, Tadayoshi Kohno, and Henry M Levy. Computer security and the modern home. *Communications of the ACM*, 2013.

[40] Sheng Ding, Jin Cao, Chen Li, Kai Fan, and Hui Li. A novel attribute-based access control scheme using blockchain for IoT. *IEEE Access*, 2019.

[41] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for iot security and privacy: The case study of a smart home. In *PerCom workshops*. IEEE, 2017.

[42] Sofia Dutta, Sai Sree Laya Chukkapalli, Madhura Sulgekar, Swathi Krithivasan, Prajit Kumar Das, and Anupam Joshi. Context sensitive access control in smart home environments. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2020.

[43] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011), 2011.

[44] Earlence Fernandes, Jaeyeon Jung, and Atul Prakash. Security analysis of emerging smart home applications. In *2016 IEEE symposium on security and privacy (SP)*. IEEE, 2016.

[45] Earlence Fernandes, Justin Paupore, Amir Rahmati, Daniel Simionato, Mauro Conti, and Atul Prakash. Flowfence: Practical data protection for emerging IoT application frameworks. In *25th {USENIX} security symposium*, 2016.

[46] David F Ferraiolo, John F Barkley, and D Richard Kuhn. A role-based access control model and reference implementation within a corporate intranet. *TISSEC*, 2(1), 1999.

[47] David F Ferraiolo, Larry Feldman, Gregory A Witte, et al. Exploring the next generation of access control methodologies. 2016.

[48] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *TISSEC*, 2001.

[49] Philip WL Fong, Mohd Anwar, and Zhen Zhao. A privacy preservation model for facebook-style social network systems. In *ESORICS*. Springer, 2009.

[50] Gartner. Gartner says the internet of things will transform the data center. `http://www.gartner.com/newsroom/id/2684616`, 2014.

[51] Dimitris Geneiatakis, Ioannis Kounelis, Ricardo Neisse, Igor Nai-Fovino, Gary Steri, and Gianmarco Baldini. Security and privacy issues for an IoT based smart home. In *2017 40th MIPRO*. IEEE, 2017.

[52] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, 2006.

[53] Jorge Granjal, Edmundo Monteiro, and Jorge Sá Silva. Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Comm. Surv. & Tutorials*, 2015.

[54] Zhang Guoping and Gong Wentao. The research of access control based on UCON in the internet of things. *Journal of Software*, 2011.

[55] Deepti Gupta, Smriti Bhatt, Maanak Gupta, Olumide Kayode, and Ali Saman Tosun. Access control model for google cloud IoT. In *2020 IEEE 6th Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2020.

[56] Maanak Gupta, James Benson, Farhan Patwa, and Ravi Sandhu. Dynamic groups and attribute-based access control for next-generation smart cars. In *Ninth ACM Conference on Data and Application Security and Privacy*, 2019.

[57] Maanak Gupta and Ravi Sandhu. 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*, 2018.

[58] Shabnam Mohammad Hasani and Nasser Modiri. Criteria specifications for the comparison and evaluation of access control models. *International Journal of Computer Network and Information Security*, 2013.

[59] Weijia He, Maximilian Golla, Roshni Padhi, Jordan Ofek, Markus Dürmuth, Earlence Fernandes, and Blase Ur. Rethinking access control and authentication for the home internet of things (IoT). In *USENIX Security 18*, 2018.

[60] Debra S Herrmann. *Using the Common Criteria for IT security evaluation*. CRC Press, 2002.

[61] Kashmir Hill. Baby monitor hack could happen to 40,000 other foscam users. `https://www.forbes.com/sites/kashmirhill/2013/08/27/baby-monitor-hack-could-happen-to-40000-other-foscam-users/613ec55458b5`, 2013.

[62] Grant Ho, Derek Leung, Pratyush Mishra, Ashkan Hosseini, Dawn Song, and David Wagner. Smart locks: Lessons for securing commodity internet of things devices. In *ASIA CCS '16*. ACM, 2016.

[63] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Comp.*, 2015.

[64] E. V. Horn J. Dennis. Programming semantics for multiprogrammed computations. In *Comm. of the ACM*, 1966.

[65] Anshul Jain and Tanya Singh. Security challenges and solutions of IoT ecosystem. In *Information and communication technology for sustainable development*. Springer, 2020.

[66] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *IFIP Annual Conf. on Data and App. Sec.*, 2012.

[67] Jia Jindou, Qiu Xiaofeng, and Cheng Cheng. Access control method for web of things based on role and sns. In *CIT 2012*. IEEE, 2012.

[68] Sun Kaiwen and Yin Lihua. Attribute-role-based hybrid access control in the internet of things. In *APWeb*. Springer, 2014.

[69] Michael S Kirkpatrick, Maria Luisa Damiani, and Elisa Bertino. Prox-rbac: a proximity-based spatially aware rbac. In *Proceedings of the 19th ACM SIGSPATIAL GIS*, 2011.

[70] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 2010.

[71] Antonio La Marra, Fabio Martinelli, Paolo Mori, and Andrea Saracino. Implementing usage control in internet of things: A smart home use case. In *2017 IEEE Trustcom/BigDataSE/ICESS*. IEEE, 2017.

[72] Aliaksandr Lazouski, Gaetano Mancini, Fabio Martinelli, and Paolo Mori. Usage control in cloud systems. In *ICITST*. IEEE, 2012.

[73] In Lee and Kyoochun Lee. The internet of things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4), 2015.

[74] J. Liu, Y. Xiao, and C. Chen. Authentication and access control in the internet of things. In *2012 32nd Int. Con. on Dist. Comp. Sys. Workshops*. IEEE, 2012.

[75] Ankur Lohachab et al. Next generation computing: Enabling multilevel centralized access control using ucon and capbac model for securing iot networks. In *IC3IoT*. IEEE, 2018.

[76] Saurav Malani, Jangirala Srinivas, Ashok Kumar Das, Kannan Srinathan, and Minho Jo. Certificate-based anonymous device access control scheme for iot environment. *IEEE Internet of Things Journal*, 2019.

[77] Fabio Martinelli, Christina Michailidou, Paolo Mori, and Andrea Saracino. Too long, did not enforce: a qualitative hierarchical risk-aware data usage control model for complex policies in distributed environments. In *CPSS '18*. ACM, 2018.

[78] Tara Matthews, Kathleen O'Leary, Anna Turner, Manya Sleeper, Jill Palzkill Woelfer, Martin Shelton, Cori Manthorne, Elizabeth F Churchill, and Sunny Consolvo. Stories from sur-

vivors: Privacy & security practices when coping with intimate partner abuse. In *CHI'17*. ACM, 2017.

[79] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. A survey of role mining. *ACM Computing Surveys*, 2016.

[80] Philipp Morgner, Stephan Mattejat, and Zinaida Benenson. All your bulbs are belong to us: Investigating the current state of security in connected lighting systems. *CoRR*, 2016.

[81] Andrew Mutsvangwa, B Nleya, and Bakhe Nleya. Secured access control architecture consideration for smart grids. In *IEEE PES PowerAfrica*, 2016.

[82] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `\T1\ textquotedblrighthttp://bitcoin.org/bitcoin.pdf`, 2008.

[83] Michele Nitti, Virginia Pilloni, Giuseppe Colistra, and Luigi Atzori. The virtual object as a major element of the internet of things: a survey. *IEEE Communications Surveys & Tutorials*, 18(2):1228–1240, 2015.

[84] Oscar Novo. Blockchain meets IoT: An architecture for scalable access management in IoT. *IEEE IoT Journal*, 2018.

[85] Mark Mbock Ogonji, George Okeyo, and Joseph Muliaro Wafula. A survey on privacy and security of internet of things. *Computer Science Review*, 2020.

[86] Temitope Oluwafemi, Tadayoshi Kohno, Sidhant Gupta, and Shwetak Patel. Experimental security analyses of non-networked compact fluorescent lamps: A case study of home automation security. In *LASER 2013*, 2013.

[87] Sylvia Osborn, Ravi Sandhu, and Qamar Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 3(2):85–106, 2000.

[88] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In *Europe and MENA Coop. Adv. in Inf. and Comm. Tech.* Springer, 2017.

[89] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access control in the internet of things: Big challenges and new opportunities. *Comp. NW*, 112, 2017.

[90] Jaehong Park. *Usage control: A unified framework for next generation access control*. PhD thesis, George Mason University, 2003.

[91] Jaehong Park, Dang Nguyen, and Ravi Sandhu. A provenance-based access control model. In *2012 Tenth Annual International Conference on Privacy, Security and Trust*, pages 137–144. IEEE, 2012.

[92] Jaehong Park and Ravi Sandhu. Towards usage control models: beyond traditional access control. In *SACMAT '02*. ACM, 2002.

[93] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, February 2004.

[94] Jing Qiu, Zhihong Tian, Chunlai Du, Qi Zuo, Shen Su, and Binxing Fang. A survey on access control in the age of internet of things. *IEEE Internet of Things Journal*, 2020.

[95] Amir Rahmati, Earlence Fernandes, Kevin Eykholt, and Atul Prakash. Tyche: A risk-based permission model for smart homes. In *2018 IEEE Cybersecurity Development (SecDev)*, pages 29–36. IEEE, 2018.

[96] Qasim Mahmood Rajpoot, Christian Damsgaard Jensen, and Ram Krishnan. Integrating attributes into role-based access control. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 242–249. Springer, 2015.

[97] Sowmya Ravidas, Alexios Lekidis, Federica Paci, and Nicola Zannone. Access control in internet-of-things: A survey. *Journal of Network and Computer Applications*, 2019.

[98] Erik Rissanen. extensible access control markup language (xacml) version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`, 2013.

[99] Ravi Sandhu, David Ferraiolo, Richard Kuhn, et al. The nist model for role-based access control: towards a unified standard. In *ACM workshop on Role-based access control*, 2000.

[100] Ravi S Sandhu. Role-based access control. In *Advances in computers*, volume 46, pages 237–286. Elsevier, 1998.

[101] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Comp.*, 1996.

[102] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *IEEE communications magazine*, 32(9):40–48, 1994.

[103] Jayasree Sengupta, Sushmita Ruj, and Sipra Das Bit. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. *Journal of Network and Computer Applications*, 2020.

[104] Arbia Riahi Sfar, Enrico Natalizio, Yacine Challal, and Zied Chtourou. A roadmap for security challenges in the internet of things. *Digital Communications and Networks*, 2018.

[105] Kewei Sha, T Andrew Yang, Wei Wei, and Sadegh Davari. A survey of edge computing-based designs for IoT security. *Digital Communications and Networks*, 2020.

[106] Hai-bo Shen and Fan Hong. An attribute-based access control model for web services. In *2006 Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)*, pages 74–79. IEEE, 2006.

[107] Sabrina Sicari, Alessandra Rizzardi, Luigi Alfredo Grieco, and Alberto Coen-Porisini. Security, privacy and trust in internet of things: The road ahead. *Computer networks*, 2015.

[108] Amit Kumar Sikder, Leonardo Babun, Z Berkay Celik, Abbas Acar, Hidayet Aksu, Patrick McDaniel, Engin Kirda, and A Selcuk Uluagac. Multi-user multi-device-aware access control system for smart home. *arXiv preprint arXiv:1911.10186*, 2019.

[109] Vijay Sivaraman, Dominic Chan, Dylan Earl, and Roksana Boreli. Smart-phones attacking smart-homes. In *WiSec '16*. ACM, 2016.

[110] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, Luciana Moreira Sá de Souza, and Vlad Trifa. Soa-based integration of the internet of things in enterprise services. In *ICWS*. IEEE Comp. Soc. Press, 2009.

[111] Statista. Forecast market size of the global smart home market from 2016 to 2022 (in billion u.s. dollars). `https://www.statista.com/statistics/682204/global-smart-home-market-size/`, 2019.

[112] Madiha Tabassum, Jess Kropczynski, Pamela Wisniewski, and Heather Richter Lipford. Smart home beyond the home: A case for community-based access control. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020.

[113] Abhijeet Thakare, Euijong Lee, Ajay Kumar, Valmik B Nikam, and Young-Gab Kim. PARBAC: Priority-attribute-based rbac model for Azure IoT cloud. *IEEE Internet of Things Journal*, 2020.

[114] Aaron Tilley. How a few words to Apple's Siri unlocked a man's front door. `http://www.forbes.com/sites/aarontilley/2016/09/21/apple-homekit-siri-security`, 2016.

[115] M. Tripunitara and N. Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security*, 15:231–272, 02 2007.

[116] Blase Ur, Jaeyeon Jung, and Stuart Schechter. The current state of access control for smart devices in homes. In *HUPS*, 2013.

[117] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM workshop on Formal methods in security engineering*, pages 45–55, 2004.

[118] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014), 2014.

[119] Yuanpeng Xie, Hong Wen, Jinsong Wu, Yixin Jiang, Jiaxiao Meng, Xiaobin Guo, Aidong Xu, and Zewu Guan. Three-layers secure access control for cloud-based smart grids. In *IEEE 82nd VTC2015-Fall*. IEEE, 2015.

[120] Hongyang Yan, Yu Wang, Chunfu Jia, Jin Li, Yang Xiang, and Witold Pedrycz. Iot-fbac: Function-based access control scheme using identity-based encryption in iot. *Future Generation Computer Systems*, 2019.

[121] Ning Ye, Yan Zhu, Ru-chuan Wang, Reza Malekian, and Lin Qiao-Min. An efficient authentication and access control scheme for perception layer of internet of things. *Applied Math. & Inf. Sciences*, 2014.

[122] Guoping Zhang and Jiazheng Tian. An extended role based access control model for the internet of things. In *2010 ICINA*. IEEE, 2010.

[123] Xinwen Zhang, Francesco Parisi-Presicce, Ravi Sandhu, and Jaehong Park. Formal model and policy specification of usage control. *TISSEC*, 2005.

[124] Yuanyu Zhang, Mirei Yutaka, Masahiro Sasabe, and Shoji Kasahara. Attribute-based access control for smart cities: A smart contract-driven framework. *IEEE Internet of Things Journal*, 2020.

[125] Yunpeng Zhang and Xuqing Wu. Access control in internet of things: A survey. *arXiv preprint arXiv:1610.01065*, 2016.

VITA

Safwa Ameer was born in Khartoum, Sudan, in 1991. In August 2012, she earned her bachelor's degree in electrical and electronic engineering from University of Khartoum, Khartoum, Sudan. After completing her engineering, she worked for two years as a Software Integration Engineer in Khartoum at Ericsson telecommunication company. In Fall 2015, Safwa started pursuing her master's degree at University of Texas at San Antonio. In Fall 2016 she started doing her thesis research under the supervision of Dr. Mathew Gibson. Safwa received her Master of Computer Science from The University of Texas at San Antonio in December 2017. Safwa entered the doctoral program in the Department of Computer Science at the University of Texas at San Antonio in Fall 2018. She joined the Institute for Cyber Security (ICS), and started doing research under the supervision of Dr. Ravi Sandhu. Her research area is IoT Access Control systems mainly focused in smart home IoT.