ATTRIBUTE-BASED ACCESS AND COMMUNICATION CONTROL MODELS FOR CLOUD AND CLOUD-ENABLED INTERNET OF THINGS

by

SMRITI BHATT, M.S.

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 IN COMPUTER SCIENCE

COMMITTEE MEMBERS:
Ravi Sandhu, Ph.D., Chair
Murtuza Jadliwala, Ph.D.
Palden Lama, Ph.D.
Gregory White, Ph.D.
Rohit Valecha, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Sciences
Department of Computer Science
August 2018

DEDICATION

I would like to dedicate this dissertation to my mom Mrs. Sarita Bhatt, my dad Mr. Yagya Raj Bhatt, and my brother Mr. Paras Bhatt for their tremendous love and support. I would also like to dedicate it to Mr. Pankaj Chhetri, my beloved, who encouraged me to pursue my Ph.D. in the first place and has been patiently supporting and motivating me throughout my journey.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Dr. Ravi Sandhu for his exceptional guidance and continuous support throughout my Ph.D. He has always encouraged me to think critically and shape my ideas in the best way possible. He has motivated me and inspired me to give my best in research as well as in life. I am and will always be thankful to him for his wisdom, knowledge, and experience that he endowed to me during my doctoral studies, which have helped me to evolve as a focused researcher and will help me in my professional growth and career.

I would like to thank Dr. Gregory White, Dr. Palden Lama, Dr. Murtuza Jadliwala, and Dr. Rohit Valecha for their time, knowledge, and valuable insights in organizing this dissertation.

I would also like to thank Mr. Farhan Patwa for his knowledge, support, and motivation throughout my research journey at UTSA.

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. I am thankful to all my friends, family members, and fellows at UTSA and beyond UTSA for providing me a strong intellectual and personal support system during my doctoral studies.

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 2018

ATTRIBUTE-BASED ACCESS AND COMMUNICATION CONTROL MODELS FOR CLOUD AND CLOUD-ENABLED INTERNET OF THINGS

Smriti Bhatt, Ph.D.
The University of Texas at San Antonio, 2018

Supervising Professor: Ravi Sandhu, Ph.D.

The essence of Attribute-Based models lies in their nature of employing attributes of various entities for controlling different aspects in a system, as defined by customized policies based on the model's objectives and application domain. In Attribute-Based Access Control (ABAC), a subject's (e.g., a user's) access to different objects (e.g., files, databases) or to subjects (e.g., other users in Online Social Networks) is secured based on the attributes of subjects and objects. ABAC controls access to data and information stored in a system by abstracting them in the form of protected objects or resources. Due to its object focused approach, ABAC is insufficient to control communications occurring in the form of streaming data and information sharing among different system components. There is some literature on controlling communications using ABAC; however, there is lack of focused treatment of Attribute-Based Communication Control (ABCC).

In today's world, two pervasive application domains are Cloud Computing and the Cloud-Enabled Internet of Things (CE-IoT). In these rapidly evolving domains, security and privacy of data and information at rest and in motion is at considerable risk at all times from unauthorized actors and malicious attackers. It is crucial to appropriately address security and privacy concerns in these two emerging domains by conducting fundamental research on specialized ABAC and ABCC models for Cloud and CE-IoT, which is currently lacking in the academic literature.

This dissertation investigates, develops, and demonstrates ABAC and ABCC models in four different contexts concerning Cloud Computing and CE-IoT. First, it develops formal ABAC models with user attributes, group attributes, and group and attribute hierarchies, viz. User-Attribute Enhanced OSAC (UAE-OSAC) model for OpenStack, and restricted Hierarchical Group and Attribute-Based Access Control (rHGABAC) model. It demonstrates enforcement of these

models utilizing unified attribute-based access control tool, the Policy Machine (PM), developed by National Institute of Standards and Technology (NIST), augmented with the Authorization Engine (AE) developed in this research.

Second, it investigates a real-world CE-IoT architecture, the AWS IoT, recently introduced by Amazon Web Services (AWS). It then develops an abstract access control model for AWS IoT known as AWS-IoTAC, based on the earlier published AWS Access Control (AWSAC) model. In contrast to AWS's policy-based approach, this dissertation identifies the need for an attribute-based approach for fine-grained authorizations in IoT and proposes ABAC enhancements to the AWS-IoTAC model. A Smart Home use case is implemented in AWS IoT to demonstrate the model and proposed ABAC enhancements.

Third, it enhances the Access Control Oriented (ACO) architecture for IoT motivated by a Wearable IoT (WIoT) use case, called the EACO architecture. It then develops an Access Control (AC) framework to comprehensively capture different types of accesses and communications within the EACO architecture for CE-IoT.

Fourth, this dissertation introduces a novel concept of Attribute-Based Communication Control (ABCC) and develops a general conceptual ABCC model. It then proposes a formal ABCC model to control data flow and enforce privacy policies between the edge IoT network and the Cloud in the context of CE-IoT. It demonstrates a real-world realization of this model using a WIoT use case and a proof-of-concept implementation employing the AWS IoT and its edge computing service.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

With the commercialization of the Internet in the 1990s and its continued advancements, various technological ideas and developments, once considered possible only as science fiction, are shaping our present and future today. One such emerging technology, supported by the ubiquitous Internet, is the Internet of Things (IoT) where all the things around us are becoming smarter and changing the way of our lives. Another pervasive technology today is Cloud Computing whose backbone is also the Internet.

With an expected number of more than 20 billion connected devices by 2020 [10], there is an inevitable need for a well-established architecture with virtually unlimited capabilities (e.g., storage, computation, analytics) to support IoT for its ongoing and future success in a sustainable manner. The integration of Cloud and IoT has been recently suggested in the literature [33, 37, 44, 45, 79, 85, 86, 90, 91]. It has also been adopted in the industry [3, 7, 12] to support the IoT architecture comprising of resource-constrained smart devices. Several terminologies are used to refer this integration, such as *Cloud-Supported IoT*, *Cloud-Assisted IoT*, and *Cloud-Enabled IoT* [35]. This dissertation uses the term Cloud-Enabled Internet of Things (CE-IoT).

The integration of these two broad domains raises many security and privacy concerns, since a large attack surface, including both Cloud and IoT vulnerabilities, is exposed to the attackers. At the same time, there is a vast amount of data and information continuously generated, stored, and shared among different components in Cloud Computing and CE-IoT architectures, which is at constant risk from malicious users and attackers.

In Cloud and CE-IoT, security and privacy of data and information both at rest and in motion are crucial. For example, in IoT domains like Medical IoT (M-IoT) and Wearable IoT (WIoT), devices are directly associated with the users. The user data and information (e.g., their personal information and behavior patterns) gathered by these devices and shared with other components in these domains are highly privacy-sensitive. This dissertation focuses on securing access and communication between various components against unauthorized and malicious entities and attackers

in Cloud and CE-IoT. The technical approach of this dissertation towards addressing security and privacy issues in these two emerging domains is to study, develop, and implement attribute-based access and communication control models, specially designed for real-world platforms pertinent to these domains.

1.1 Motivation

Access Control (AC) in general refers to control of access to a protected object (e.g., file, folder, and database) by an authorized subject (e.g., user) in a system. The object here corresponds to the data stored or a resource in a server, a system, or an application. While controlling access is critical, it is also necessary to secure communications between different components or entities in a system. In Cloud and CE-IoT, there are billions of users and devices, and the data and information associated with them are highly sensitive and exposed to cybersecurity threats in both the physical and virtual environment. These architectures comprise several entities and a network that enables communication between these entities.

In a CE-IoT architecture, there are various types of communications and interactions between different components. For example, in a wearable IoT scenario, there are wearable devices, associated with a specific user, at the edge network which are connected to a gateway that communicates to their associated virtual objects (digital counterparts of the physical devices) [82] residing in the Cloud computing platform. In such architectures, the data and information stored and exchanged between several components are at considerable risk at all times and need to be secured with appropriate access and communication control mechanisms. Due to the dynamic nature of Cloud and CE-IoT, a flexible attribute-based approach is employed in this dissertation to develop access and communication control models for Cloud and CE-IoT.

Attribute-Based Access Control (ABAC) [63,64,67] models provide flexible access control and authorization mechanism based on the attributes (or properties) of entities, such as users, subjects, and objects, in any application or system. The history of ABAC dates back to over two decade. However, ABAC research in academia has gained momentum in recent years with the development

of several ABAC models with basic and additional capabilities [43,65,67,87,101,103,112]. ABAC models have also been applied in administrative context for controlling administrator's accesses on model entities, such as users, objects, subjects, roles [36, 58, 80, 81]. While some research on ABAC for Cloud and IoT has been conducted, there are so far no unified models which could be widely applied in these domains in industry. Significant research on ABAC models customized and designed for Cloud and CE-IoT platforms along with their demonstration in real-world scenarios is necessary for their wide-adoption in the industry.

Moreover, ABAC focuses on securing access to static objects residing or stored in a machine. However, the security of communication occurring between two components in a system, where data and information are continuously flowing from one point to the other, is essential to defend against malicious users and attackers. While some efforts [47,48] have been made to secure access to communications procedures by modeling them as resources in the access control mechanism in IoT. These efforts focus on a particular IoT protocol (e.g., MQTT), and thus cannot be generalized to represent all types of communications in IoT. Besides, they do not utilize an attribute-based approach in controlling the communications.

Unlike well-developed access control models, such as DAC [94], MAC [94], RBAC [54,92,93], and ABAC [63, 64, 67]) for securing accesses to protected objects, there is a lack of emphasis on formal communication control models for controlling communications and data flow between two components in a system. Therefore, this dissertation proposes a novel concept of Attribute-Based Communication Control (ABCC) for securing communications and data flow between different entities. Since attribute-based communication control is a new concept, currently a basic understanding of ABCC models is lacking in the literature. Research on ABCC and its characteristics is necessary to develop a conceptual ABCC model. This dissertation contributes towards formalizing the ABCC model and defining its components and relationship with the ABAC model.

ABCC models can be employed to defend against security attacks (e.g., eavesdropping, Denial of Service (DoS)) and enforce user-centric privacy policies in the CE-IoT architecture. For example, in a WIoT scenario, an attacker could eavesdrop on the communication between the devices

at edge network and Cloud to gather user data and create a profile based on their information and behavior to gain sensitive personal information. Another example is that of a user concerned about privacy and does not want to send the user's physiological and environment data (e.g., number of steps, location) to the Cloud but is willing to send such data to the edge device where it will not be persistently stored.

In distributed Cloud and IoT architectures, where a tremendous amount of data and information is continuously being generated, stored, and shared among various entities, secure access control and communication control models need to be studied and developed for ensuring security and privacy of user data and information.

1.2 Problem Statement

While Attribute-Based Access Control (ABAC) has been applied to address access control and authorization in Cloud and IoT, there is still a significant gap in theoretical ABAC models and their application in real-world Cloud and CE-IoT platforms. On the other hand, there is fundamental lack of knowledge and academic literature with respect to Attribute-Based Communication Control (ABCC), a novel concept introduced in this dissertation to secure communication and data flow in CE-IoT. A conceptual ABCC model, its characteristics, and its relationship with ABAC need to be studied and defined for facilitating the development of concrete ABCC models for real-world CE-IoT applications.

1.2.1 Thesis Statement

A flexible attribute-based approach can be utilized to address security and privacy issues in the dynamic and rapidly progressive Cloud Computing and CE-IoT architectures. A detailed exploration of ABAC and ABCC, their formal models, and implementation in different contexts concerning Cloud Computing and CE-IoT can ultimately strengthen the access, authorization, and communication framework in these domains.

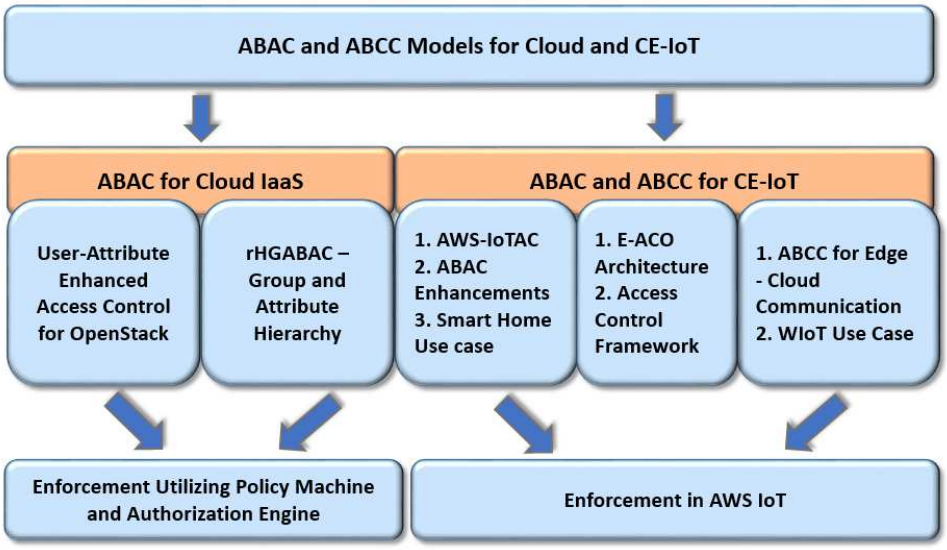


Figure 1.1: Overview of Contributions

1.3 Scope and Assumption

The scope of this dissertation is to develop appropriate Attribute-Based Access Control (ABAC) and Attribute-Based Communication Control (ABCC) models and demonstrate their applicability in Cloud Computing and CE-IoT to enhance overall security architecture of these domains. This dissertation is based on the following assumptions:

- Communications in an IoT architecture, comprising the flow of data and information between two entities or components, need to be controlled including user-driven policies.
- ABAC by itself is insufficient to secure continuously flowing data and information in distributed application domains. Hence, ABCC models are introduced.
- Edge Computing is necessary to support disruptively expanding IoT architecture.

1.4 Summary of Contributions

Figure 1.1 presents an overview of the contributions of this dissertation which can be divided into four contexts related to Cloud Computing and CE-IoT. First, ABAC models for Cloud IaaS, i.e., OpenStack [100]), second, access control model and ABAC for Amazon Web Services (AWS)

IoT [3], third, enhancements to the recently published ACO architecture for IoT [35] and Access Control Framework for CE-IoT, and fourth, ABCC for securing communications and data flow in CE-IoT. The main contributions of this dissertation are as follows:

- It develops two ABAC models in the context of Cloud Computing IaaS, first extending the OpenStack access control with user attributes, and second a more general ABAC model with additional capabilities (e.g., groups and attribute hierarchies) that can be applied in Cloud IaaS platforms. It demonstrates their applicability utilizing a novel enforcement architecture including the Policy Machine [51,52], augmented the Authorization Engine (AE) developed in this research.

- In the context of CE-IoT, it develops an access control model for AWS IoT, viz. AWS-IoTAC, and proposes ABAC enhancements for it to enable fine-grained and flexible access control mechanism in AWS IoT. A smart home use case is developed to demonstrate the application of the model and depict the benefits of ABAC enhancements.

- Motivated by a Wearable IoT (WIoT) use case, it enhances the recently published Access Control Oriented (ACO) architecture [35] and call it EACO. With respect to the EACO, it develops an Access Control framework for grouping various types of interactions in CE-IoT.

- Lastly, it introduces a novel concept of Attribute-Based Communication Control (ABCC). It builds a conceptual model for ABCC and explores its characteristics against the ABAC model. It takes a first step in the development of a set of ABCC models for the CE-IoT architecture by developing the ABCC-EC model for securing communications between the edge network and the Cloud platform.

1.5 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 including ABAC, HGABAC, OpenStack and its access control model, Policy Machine, and ACO architecture for IoT. Chapter 3 discusses two

different ABAC models which could be applied in Cloud IaaS platforms and demonstrates their applicability and feasibility in real-world scenarios. Chapter 4 presents an access control model for AWS IoT and discusses the proposed ABAC enhancements to AWS IoT. It also presents a Smart Home use case to depict the benefits of ABAC in IoT. Chapter 5 presents the enhanced ACO (EACO) architecture and develops an access control framework to categorize various interactions (accesses and communications) in CE-IoT. Chapter 6 introduces the notion of ABCC to secure different types of communications in the CE-IoT architecture. It establishes a conceptual ABCC model and describes its characteristics and relationship with ABAC, and also develops a formal ABCC model for edge to Cloud communications. Finally, Chapter 7 concludes the dissertation with a brief overview of potential future work.

CHAPTER 2: BACKGROUND

This chapter discusses fundamental concepts and background essential to comprehend the research contributions of this dissertation. It describes ABAC, a central concept in this dissertation, and reviews the HGABAC model. It also discusses OpenStack and its access control model, the Policy Machine and its architecture, AWS access control model, and the recently published ACO architecture for IoT. These topics are ordered based on their relevance to the following chapters.

2.1 Attribute-Based Access Control (ABAC)

Access control is a mechanism that determines *who* can do *what* and on *which resources*. It includes two parts: *authentication* which deals with the *who* part and *authorization* that identifies *what* authenticated users can do on *which* resources. There have been several access control models proposed and formalized in the literature. These models act as the fundamental security mechanism in numerous applications and systems. Among these different access control models, only a few have been successfully applied in real-world applications and systems.

Three most significant and widely known access control models are Discretionary Access Control (DAC) [94], Mandatory Access Control (MAC) [94], and Role-Based Access Control (RBAC) [54, 92, 93]. While each of them has their advantages, they also have weaknesses. In DAC, owners of objects control access of users to the objects. It is simple and straightforward but has inherent weaknesses, such as copying problem and trojan horses which can be easily exploited to gain unauthorized access to sensitive information. Similarly, MAC provides a more strict access control method by assigning security labels to users and object. It is designed for military applications with the focus to maintain the confidentiality of the information. Whereas, RBAC is a more flexible and administrative friendly access control model [43]. 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 [89].

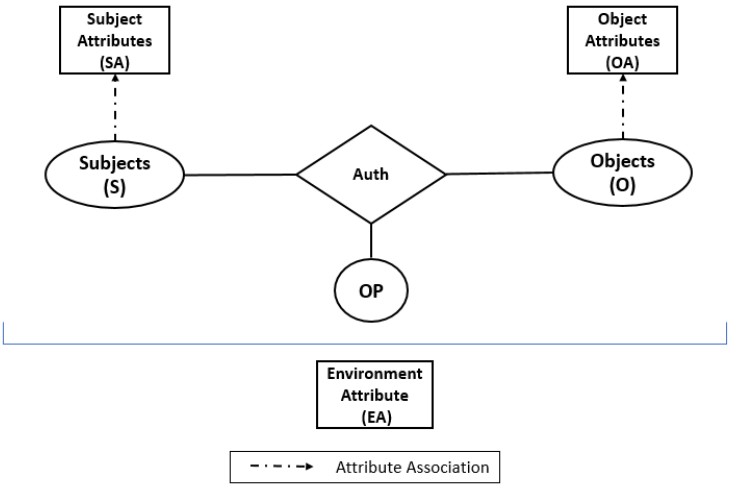


Figure 2.1: A Simple Conceptual ABAC Model (Adapted from [67])

Motivated by the limitations of the traditional access control models, attribute-based access control (ABAC) has recently received significant attention in the literature [43]. 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. ABAC authorization policies are a set of rules defined based on the attributes of subjects and objects as well as other attributes, such as contextual attributes.

The concept of ABAC has been around for over two decades in the literature with attribute-based access control models designed for specific applications [103, 108] and Attribute-Based Encryption [56, 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) [53]. A unifying formal ABAC model, known as ABACα, along with its features, components, and formal specifications is presented by Jin et al. in [67]. It also depicts that ABAC can express the traditional access control models (DAC, MAC, and RBAC) by suitably defining the attributes.

Figure 2.1 presents a simple conceptual ABAC model, adapted from [67]. The core compo-

nents of the model are *Subjects (S), Objects(O), Subject attributes (SA), Object Attributes (OA), Operations (OP),* and *an Authorization Function (Auth)*. The contextual or environmental attributes (EA) also exists in the model space and can be utilized in the access control policy based on the requirements. For example, an employee's request to access office resources is granted only during the daytime; thus here *daytime* is the environment attribute in the authorization policy.

The subjects represent individual users in a system. They are entities or processes created by and executing on behalf of the users. The objects are protected data, information, and resources, such as a database, file, a printer, etc. The subject attributes represent the properties of the subjects, such as *Name, Age, Title, etc.*, and object attributes represent the properties of the objects, such as *Type, Owner, etc.* Typically these attributes are name-value pairs. As defined in [67], an attribute is a function with a range of values that takes an entity (e.g., user, object) as the input and returns a value from its range. The range of an attribute is a finite set of atomic values.

The attributes are of two types: *atomic-valued* which returns a single value for an attribute, and *set-valued* which returns a set of values for an attribute. The operations could be simply *read* and *write*. The authorization function evaluates the authorization policy and returns *Allow* or *Deny* for an access request. The nature and concrete details of these components are implementation specific and depend on the discretion of the system administrator. Jin et al. provide a detailed formal definition of ABAC components along with a policy specification language in [67].

The ABAC models can be classified into two classes based on the type of authorization policy specification techniques which are *Logical-formula Authorization Policy (LAP)* and *Enumerated Authorization Policy (EAP)*. In LAP, predicate logic and logical operators (e.g., AND, OR) are used to define the ABAC authorization policy including attributes and their values. It provides the flexibility to define complex authorization policies in a simple and easy manner. However, policies in LAP are heterogeneous since there are no constraints on the size and structure of policy. Thus, it difficult and cumbersome for a system administrator to manage, review and update logical formula policies [42]. Whereas in EAP, the authorization policy is specified as enumerated tuples involving subject attributes and object attributes. It is a set of one or more tuples where the structure of

policies is homogenous. Therefore, EAP provides administrative scalability to review and update policies. EAP policies can be updated by adding or removing tuples. However, EAP has its disadvantages, such as large policy size due to enumerations of each condition in the policy and support for limited operators [42]. Biswas and Sandhu present a detailed comparison of LAP and EAP in [42]. An instance of EAP-ABAC is the Policy Machine [51, 52] developed by NIST, which has been utilized in the following chapter to present a unique enforcement architecture for enforcing ABAC models.

This dissertation develops and formalizes ABAC models in different context of Cloud Computing and CE-IoT. These models include the basic ABAC structure and features discussed here, as well as introduce additional capabilities (components and relationships) based on the requirements of the specific domain. It discusses both LAP-ABAC and EAP-ABAC models based on the context of the application. An instance of EAP-ABAC model, a restricted HGABAC (*r*HGABAC), is developed and formally defined along with enforcement of the model in Chapter 3. Originally, the HGABAC model has been developed and formalized as a LAP [58, 101], which is discussed in the next section.

2.1.1 HGABAC Model

In [101], Servos and Osborn introduced user and object groups with hierarchical relationship and attributes in ABAC and proposed a formal Hierarchical Group and Attribute-Based Access Control (HGABAC) model. In the process of developing an administrative model for HGABAC, named $GURA_G$, Gupta and Sandhu presented a conceptual model of HGABAC with an alternate formalization for it based on $ABAC\alpha$ [58].

Figure 2.2 shows the conceptual HGABAC model formalized in [58]. It consists of basic ABAC components, such as **users (U), subjects (S), objects (O), user attributes (UA), object attributes (OA),** and **operations (OP)**. A user is an individual and represents actual humans interacting with a system, whereas subjects are the representation of users in the virtual environment, such as a process or a session running on behalf of the user who performs operations on an object

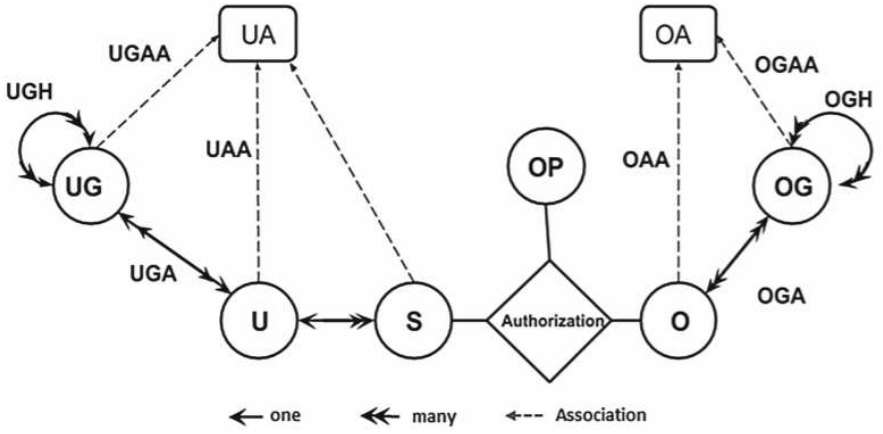


Figure 2.2: A Conceptual HGABAC Model [58]

in a system. Objects are data and information resources, such as files, databases, and applications. Operations are actions (e.g., read, write) that can be performed on the objects by the subjects. User attributes represent the properties of users and subjects, and object attributes represent the properties of the attributes. Subjects are created by users. Here, all attributes are considered to be set valued, and thus each attribute can be assigned a subset of values from the range of an attribute, represented as Range(att) [58].

User groups (UG) and **object groups (OG)** are a collection of users and objects respectively. User groups are assigned a set of user attributes, and object groups have a set of object attributes assigned to them. A user in a user group inherits attributes from that user group, and objects which are members of a specific object group inherits attributes from that group. Furthermore, these groups have partial order hierarchical relationship among them [58]. A group hierarchy is a partial order relation written as $\succeq_g$ where senior groups acquire all attribute values assigned to the groups junior to them, along with their own directly assigned attribute values. It is especially beneficial when there exist many users in a system having common characteristics. Therefore, instead of assigning same attribute values to each user, we can group the users into specific groups and assign appropriate attributes and their values to these groups. Similar is the case for objects where objects having same properties and characteristics can be grouped into one object group.

Figure 2.3 shows a simple user group hierarchy example. There are three groups *Computer*

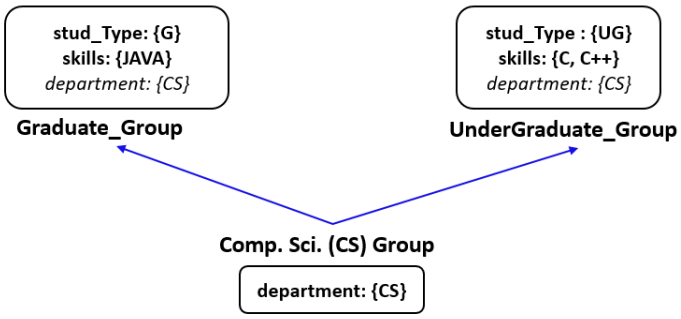


Figure 2.3: An Example of User Group Hierarchy (Adapted from [58])

Science Group, *Graduate_Group*, and *Undergraduate_Group*. Among these graduate and undergraduate are senior groups and are represented higher up whereas computer science is a junior group and is one level below the senior groups. An attribute and its values are written as $att\_name : \{Val_1, Val_2, ..., Val_n\}$. A directly assigned attribute-value is shown in bold font, and an inherited attribute-value is shown as italicized in normal font in Figure 2.3. In this example, the senior groups, *Graduate_Group* and *Undergraduate_Group*, inherit attribute *department* and its value *CS* from junior group *Computer Science Group* and also have directly assigned attributes *skills* and *stud_Type*.

The formal definitions of the model, as given in [58], are presented in Table 2.1. As discussed earlier, **U, S, O, OP, UG, OG, UA,** and **OA** are finite sets of users, subjects, objects, operations, user groups, object groups, user attributes, and object attributes respectively. For each attribute *(att)*, there is a finite range of attributes, represented as *Range(att)*. User attribute functions are represented as att_u while att_o represents the object attribute functions. A mapping of user to user groups is given by **directUg** and mapping of objects to object groups is given by **directOg**. **User Group Hierarchy (UGH)** represents a partial order relationship between user groups, written as $\succeq_{ug}$. For example, $ug_1 \succeq_{ug} ug_2$ means that ug_1 is a senior group and ug_2 is a junior group in the user group hierarchy. **Object Group Hierarchy (OGH)** represents a partial order relationship between object groups, written as $\succeq_{og}$.

The derived functions of the model include effective attributes of user groups, users, object groups, and objects, and effective attributes of the subjects. The effective attributes are derived

Table 2.1: An Alternate Formalization for HGABAC Model [58]

Basic Sets and Functions

– U, S, O, OP are finite set of users, subjects, objects and operations respectively
– UG, OG are finite set of user and object groups respectively
– UA, OA are finite set of user and object attribute functions respectively
– For each att in $\mathrm{UA} \cup \mathrm{OA}$, Range(att) is a finite set of atomic values
– For each $\mathrm{att_u}$ in UA, $\mathrm{att_u} : \mathrm{U} \cup \mathrm{UG} \rightarrow 2^{\mathrm{Range}(\mathrm{att_u})}$, mapping each user and user group to a set of values in $\mathrm{Range}(\mathrm{att_u})$
– For each $\mathrm{att_o}$ in OA, $\mathrm{att_o} : \mathrm{O} \cup \mathrm{OG} \rightarrow 2^{\mathrm{Range}(\mathrm{att_o})}$, mapping each object and object group to a set of values in $\mathrm{Range}(\mathrm{att_o})$
– directUg : $\mathrm{U} \rightarrow 2^{\mathrm{UG}}$, mapping each user to a set of user groups
– directOg : $\mathrm{O} \rightarrow 2^{\mathrm{OG}}$, mapping each object to a set of object groups
– $\mathrm{UGH} \subseteq \mathrm{UG} \times$UG, a partial order relation $\succeq_{ug}$ on UG
– $\mathrm{OGH} \subseteq \mathrm{OG} \times$OG, a partial order relation $\succeq_{og}$ on OG

Effective Attributes (Derived Functions)

– For each $\mathrm{att_u}$ in UA,
- $\mathrm{effectiveUG_{att_u}} : \mathrm{UG} \rightarrow 2^{\mathrm{Range}(\mathrm{att_u})}$, defined as $\mathrm{effectiveUG_{att_u}}(\mathrm{ug_i}) = \mathrm{att_u}(\mathrm{ug_i}) \cup (\bigcup_{\forall \mathrm{g} \in \{\mathrm{ug_j} | \mathrm{ug_i} \succeq_{ug} \mathrm{ug_j}\}} \mathrm{effectiveUG_{att_u}}(\mathrm{g}))$
- $\mathrm{effective_{att_u}} : \mathrm{U} \rightarrow 2^{\mathrm{Range}(\mathrm{att_u})}$, defined as $\mathrm{effective_{att_u}}(\mathrm{u}) = \mathrm{att_u}(\mathrm{u}) \cup (\bigcup_{\forall \mathrm{g} \in \mathrm{directUg(u)}} \mathrm{effectiveUG_{att_u}}(\mathrm{g}))$

– For each $\mathrm{att_o}$ in OA,
- $\mathrm{effectiveOG_{att_o}} : \mathrm{OG} \rightarrow 2^{\mathrm{Range}(\mathrm{att_o})}$, defined as $\mathrm{effectiveOG_{att_o}}(\mathrm{og_i}) = \mathrm{att_o}(\mathrm{og_i}) \cup (\bigcup_{\forall \mathrm{g} \in \{\mathrm{og_j} | \mathrm{og_i} \succeq_{og} \mathrm{og_j}\}} \mathrm{effectiveOG_{att_o}}(\mathrm{g}))$
- $\mathrm{effective_{att_o}} : \mathrm{O} \rightarrow 2^{\mathrm{Range}(\mathrm{att_o})}$, defined as $\mathrm{effective_{att_o}}(\mathrm{o}) = \mathrm{att_o}(\mathrm{o}) \cup (\bigcup_{\forall \mathrm{g} \in \mathrm{directOg(o)}} \mathrm{effectiveOG_{att_u}}(\mathrm{g}))$

Effective Attributes of Subjects (Assigned by Creator)

– SubUser : S → U, mapping each subject to its creator user
– For each $\mathrm{att_u}$ in UA, $\mathrm{effective_{att_u}} : \mathrm{S} \rightarrow 2^{\mathrm{Range}(\mathrm{att_u})}$, mapping of subject s to a set of values for its effective attribute $\mathrm{att_u}$. It is required that : $\mathrm{effective_{att_u}}(\mathrm{s}) \subseteq \mathrm{effective_{att_u}}(\mathrm{SubUser(s)})$

Authorization Function

For each op ∈ OP, $\mathrm{Authorization_{op}}$ (s:S, o:O) is a propositional logic formula, returning true or false and is defined using the following policy language:

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\, \mathrm{x} \in \mathrm{set}.\alpha \mid \forall\, \mathrm{x} \in \mathrm{set}.\alpha \mid \mathrm{set} \bigtriangleup \mathrm{set} \mid \mathrm{atomic} \in \mathrm{set} \mid \mathrm{atomic} \notin \mathrm{set}$
- $\bigtriangleup ::= \subset \mid \subseteq \mid \not\subseteq \mid \cap \mid \cup$
- $\mathrm{set} ::= \mathrm{effective_{att_{u_i}}}(\mathrm{s}) \mid \mathrm{effective_{att_{o_i}}}(\mathrm{o})$ for $\mathrm{att_{u_i}} \in$ UA, $\mathrm{att_{o_i}} \in$ OA
- $\mathrm{atomic} ::= \mathrm{value}$

Access Decision Function

A subject $\mathrm{s_i} \in$ S is allowed to perform an operation op ∈ OP on a given object $\mathrm{o_j} \in$ O if the effective attributes of the subject and object satisfy the policies stated in $\mathrm{Authorization_{op}}(\mathrm{s} : \mathrm{S}, \mathrm{o} : \mathrm{O})$. Formally, $\mathrm{Authorization_{op}}(\mathrm{s_i}, \mathrm{o_j})$ = True

based on the directly assigned attributes to entities (users and objects) and attribute inherited as a result of many-to-many group hierarchy (UGH and OGH). For a user, the effective attribute values of a user attribute att_u is the union of directly assigned user attribute values and attribute values inherited from all the user groups of which the user is a member. The effective attribute values of a user group $effectiveUG_{att_u}$ is the union of user attribute values directly assigned to the user group in *UG* and the attribute values inherited from all the junior user groups. Similarly, the effective attributes for objects and object groups are defined in Table 2.1.

The **SubUser** function maps a subject to the user who created this subject. The effective attributes of subjects are derived based on their creators, which in this case are specific users. Therefore, a subject's effective attribute values are a subset of its user effective attribute values. The authorization function $Authorization_{op}(s, o)$ for a specific operation *op* in the model is defined as a logical-formula based on the policy language. It determines if a subject *s* can perform operation *op* on an object *o* based on the access decision function which utilizes the effective attribute values of subject *s* and object *o* in evaluating the access decision (Allow/Deny) [58].

The conceptual HGABAC model discussed above utilizes the logical-formula authorization policy and is a LAP-ABAC model. In Chapter 3, an enumerated authorization policy (EAP) version of the HGABAC model, called restricted HGABAC (*rHGABAC*), is developed that incorporates and demonstrates the benefits of EAP-ABAC. The *rHGABAC* model is enforced utilizing the Policy Machine (PM) and the Authorization Engine (AE), a proof-of-concept implementation developed in this dissertation.

2.2 OpenStack

OpenStack is a widely used open source cloud computing platform. It provides a robust IaaS platform for building public, private or hybrid clouds. It is a rapidly evolving application which is developed and maintained by a vibrant community of developers from more than 200 world-leading organizations. It is formally defined as follows in [24]:

"OpenStack software controls large pools of compute, storage, and networking resources through-

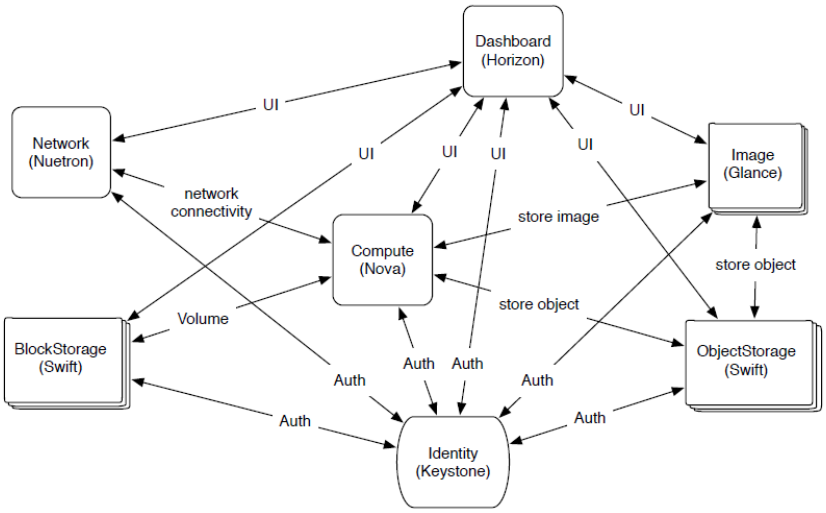


Figure 2.4: The OpenStack Architecture [24]

out a datacenter, managed through a dashboard or via the OpenStack API. OpenStack works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure."

Figure 2.4 depicts the OpenStack architecture. OpenStack consists of various services such as compute (*Nova*) compute service, image (*Glance*) service, identity (*Keystone*), block storage (*Cinder*), Dashboard (*Horizon*), object storage (*Swift*), and networking (*Neutron*). *Horizon* is the web-based dashboard that allows users to access all the services through a GUI; however, there is also a command line interface (CLI) for users to interface with each of the services.

Nova is a compute service and allows users to create virtual machines, and *Swift* provides data storage via swift objects. Similarly, *Cinder* provides block storage attached to virtual machines as a storage volume. *Glance* provides users with images (e.g., OS, software, configurations, etc.) that are used in instantiating virtual machines. *Neutron* provides networking services to the users allowing them to network virtual machines using virtual routers. *Keystone* is the identity service which manages the overall security of OpenStack including authentication and authorization [24].

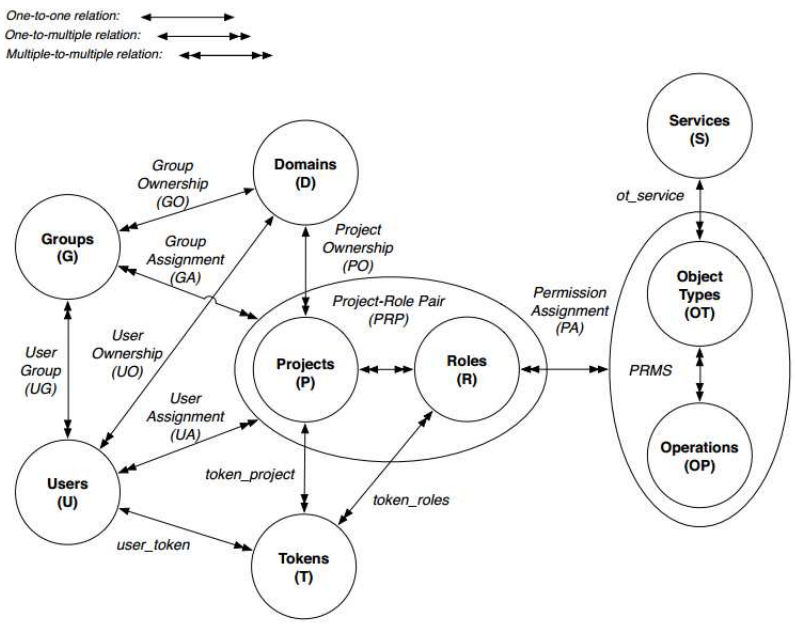


Figure 2.5: OpenStack Access Control (OSAC) Model [105]

2.2.1 OpenStack Access Control (OSAC) Model

Tang and Sandhu [105] developed an access control model for OpenStack, known as OSAC, based on the OpenStack Identity API v3 and Havana release. Figure 2.5 presents the core OSAC model. It consists of nine entities: *users, groups, projects, domains, roles, services, object types, operations*, and *tokens*. *Users* are individuals authenticated to access cloud resources, *groups* are a set of users, and *projects* are resource containers through which users get access to specific cloud resources such as virtual machines (VMs), storage, etc. The *Domain* is a higher level concept that represents a tenant of the cloud service provider (CSP). The projects in a domain represent the administrative boundary of its users and groups. They allow tenants to segment their resources and to manage their users' scope of access to those resources [116].

Roles are global entities used to associate users with any of the projects inside a domain. It specifies the access levels of users to services in specific projects in a given domain. *Permissions* are assigned to role-project pairs and are used to specify access levels of users to services in specific projects, with specific roles. Role-permission assignments are defined by a cloud administrator. *Object types* are different types of resources in cloud services such as virtual machines (VMs),

images, swift-objects, etc. *Operations* are access methods on these object types owned by *services* in the cloud. An object type and operation pair defines actions which can be performed by end users on cloud services and resources. Each authenticated user receives a *token* from the identity service Keystone, which represents the scope of resources that a user is allowed to access. A token is equivalent to a subject and has information about the user, its roles in specific projects and its associated domain [116].

OpenStack access control utilizes the role-based approach. It is an open source Cloud IaaS platform that is widely being used to create different types of cloud architectures: *Public Cloud, Private Cloud,* and *Hybrid Cloud* in industry and academia. With such broad application of OpenStack in the real-world, the customers have already started to realize the weaknesses of RBAC, i.e., role explosion and role-permission explosion. Therefore, a flexible access control mechanism, such as ABAC, needs to be explored for OpenStack to enhance its access control framework. At the same time, it is a widely used platform, thus replacing RBAC completely in OpenStack is not feasible. Chapter 3 presents a simplified version of the core OSAC model along with formal definitions. It then proposes an ABAC extension for OpenStack, the User-attribute Enhanced OSAC (UAE-OSAC) model, while keeping its RBAC authorization framework intact. The UAE-OSAC model is enforced in OpenStack using the Policy Machine (PM) and our proof-of-concept implementation, the Authorization Engine (AE).

2.3 The Policy Machine

The Policy Machine (PM) [51, 52] is a general-purpose attribute-based access control framework developed by the National Institute of Standards and Technology (NIST). It is a reference implementation for the Next Generation Access Control (NGAC), an emerging ANSI/INCITS standard being developed by NIST. PM is an open source application. The first version of PM/NGAC implementation is known as Harmonia-1.5. However, recently a new version, Harmonia-1.6 has been developed with an enhanced architecture (i.e., Object-Oriented Software Architecture) including MySQL support [15]. As per NIST first public release of the PM, it is defined as follows [14].

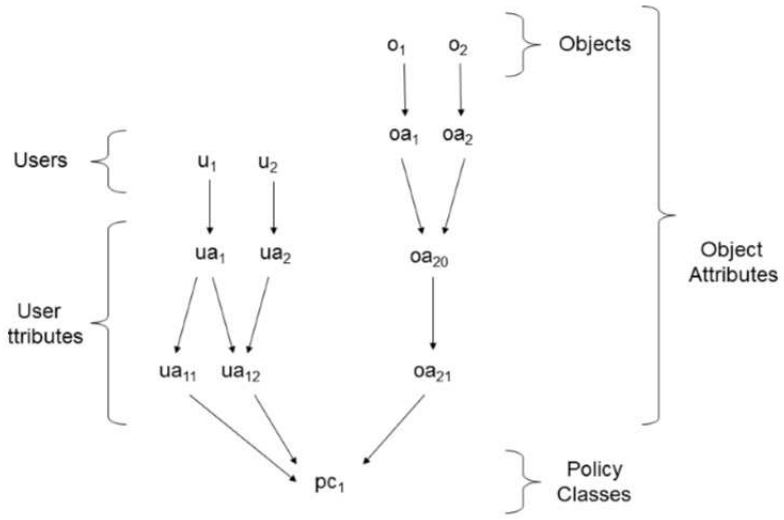


Figure 2.6: A Simplified Policy Element Diagram [52]

"The Policy Machine is an access control mechanism that comprises: (1) Access control data used to express access control policies and deliver capabilities of data services to perform operations on objects; (2) a set of administrative operations for configuring the access control; and (3) a set of functions for enforcing policy on requests to execute operations on objects and for computing access decisions to accommodate or reject those requests based on the current state of the access control data."

This dissertation uses the Harmonia 1.5 release, and the PM concepts and architectures discussed here are relevant to this release [51, 52]. The PM can express and enforce arbitrary, organization specific, attribute-based access control policies. It is a mechanism to define access control policies in terms of a standardized and generic set of relations and functions that can be reused. The primary objective of PM is to provide a unifying framework to support a wide range of attribute-based policies or policy combinations.

The PM has eight core elements or entities: **users, objects, user attributes, object attributes, operations, processes, access rights** and **policy classes**. The policy classes, user attributes, and object attributes are containers for policies, users, and objects respectively. The PM has four types of relations: **assignment, association, prohibition** and **obligation**, and two sets of functions: **access control decisions** and **policy enforcement**. *Assignment* relation is used to define the re-

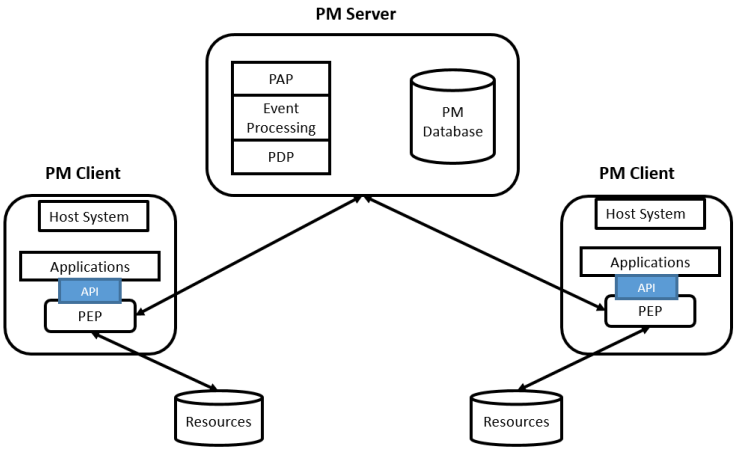


Figure 2.7: Policy Machine Architecture (Adapted from [51])

lationship between users, user attributes, objects and object attributes, and *association* relation is used for making the association between user attributes and object attributes or objects through some operations. These associations define access control policies. *Prohibition* and *obligation* relations are used to enforce constraints and restrictions on user capabilities in specific access control policies.

Through the *assignment* relation in the PM, there exists a *containment* property among PM attributes. The containment property implies that if there exist any two elements x and y such that x is assigned to (contained in) y by one or more assignment relations, then x acquires or gets all the properties and capabilities of y in addition to its own directly conferred properties [52]. PM supports hierarchical relations through the containment property. Based on the existing set of PM elements and relations, different types of access control policies (e.g., DAC, MAC, RBAC) can be specified and enforced utilizing the PM.

In PM, a policy element diagram represents a directed policy graph which comprises basic policy elements and assignments among them. Figure 2.6 depicts a simplified policy element diagram. It is an inverted graph with arrows pointing down. It shows assignments between different type of policy elements [52]. In Figure 2.6, users and objects are assigned to user attributes and object attributes respectively. User and object attributes are assigned to other user and object attributes respectively. Finally, all the elements are assigned to a policy class. Similar policy

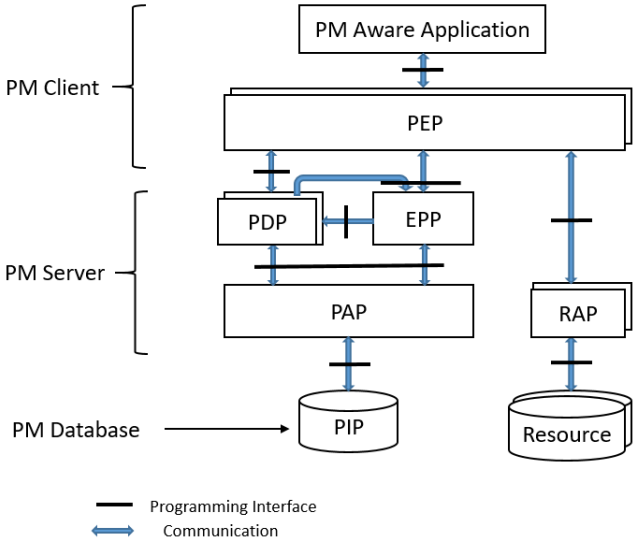


Figure 2.8: Architectural Components of the PM (Adapted from [52])

graphs are generated and discussed in depth while enforcing relevant use cases in the restricted HGABAC (*rHGABAC)* model in Chapter 3.

The general PM architecture is shown in Figure 2.7. It comprises a PM server, PM clients, and Resources repository. PM server includes a PM Database (Active Directory), a Policy Decision Point (PDP), a Policy Administration Point (PAP) and an event processing module. The PM clients are host systems where the policies are enforced. They encapsulate the application programming interfaces (API) and PM-aware applications. In current PM implementation, the Policy Enforcement Point (PEP) is implemented as a kernel simulator. The PM client or the user environment is the context in which the user's PM processes run. These processes are similar to subjects. A PM client could be an operating system, an application (e.g., a database management system), a service in a service-oriented architecture, or a virtualized environment. The resources are the repositories for different types of objects such as files, records, directories, etc. [52].

Figure 2.8 shows the architectural components of the PM, from an alternate perspective. For a general application to be PM compliant, the applications need to modified to incorporate and be able to communicate to the PEP in the PM which talks to the PDP for access control decisions. A PDP determines whether an access request made by PEP should be granted or denied as per the

policy defined in PAP. All the information regarding access control data and relations is stored in a PM database, which is a policy information point (PIP).

Policy Machine supports a rich set of capabilities for defining customized access control policies. It allows to define deny or prohibition relations and apply constraints, and also allows combining different access control policies specified in PM using the Admin tool. PM Admin tool is a GUI based tool, used to define and administer access control policies in PM by creating policy classes, users, user attributes, objects, object attributes, and setting operations sets and permissions between user attributes and object attributes and objects. All this data, information, and relations are stored in Active Directory (the PM Database) as in PM version 1.5. Users request access to objects through PM clients which in turn communicate to the PM server for access control decisions and enforce these decisions on host systems. PM follows an enumerated authorization policy (EAP) mechanism to specify attribute-based access control policies.

One of the other known attribute-based frameworks for enforcing ABAC policies is eXtensible Access Control Markup Language (XACML) [32]. It is an OASIS standard and has been used to define ABAC policies in the academia and industry. It is an attribute-based access control policy language for managing authorized access to resources. It provides an architecture with a standard set of components, such as policy administration point (PAP), policy decision point (PDP), policy information point (PIP), policy enforcement point (PEP), etc. for evaluating access control policies. It follows a logical formula ABAC policy system where predicate logic is used to specify policy rules. A formal role-centric attribute-based access control (RABAC) model has been proposed by Jin et al. [69] where the authors utilized an XACML based enforcement technique for their model.

This dissertation utilizes the PM to develop an enforcement architecture for ABAC models. The capabilities of the PM and the flexibility to specify and combine different types of access control policies make it a suitable choice for utilizing it as an enforcement architecture. On the downside of the PM, the applications using it should be aware of its structure and elements, and this adds complexity. To simplify the process and ease of use of the PM with applications as OpenStack, a proof-of-concept implementation of an authorization engine (AE) is developed in

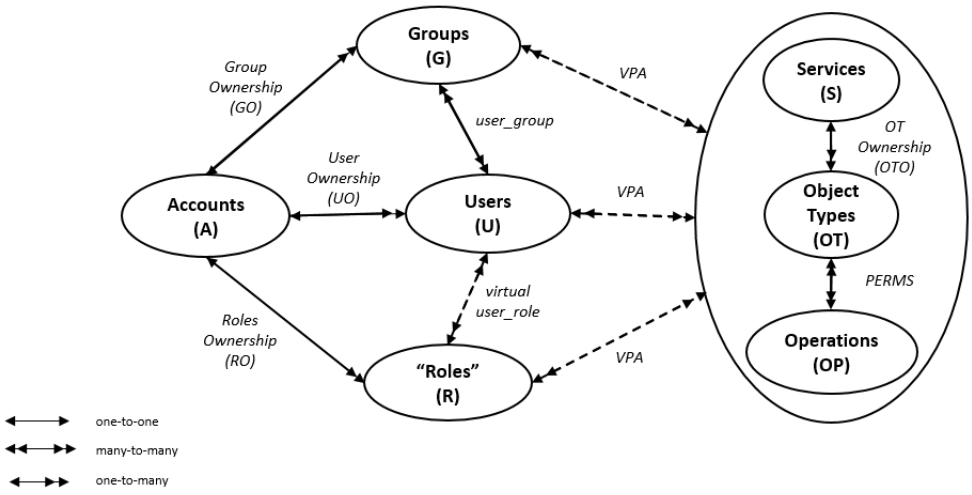


Figure 2.9: AWS Access Control within a Single Account [115]

this dissertation to facilitate interaction between the PM and applications using it. This novel enforcement architecture with PM and AE will be discussed and applied in Chapter 3.

2.4 AWS Access Control (AWSAC) Model

Amazon Web Services (AWS) is a public cloud computing platform provided by Amazon [1]. As per [1], it is defined as follows:

"Amazon Web Services (AWS) is a secure cloud services platform, offering compute power, database storage, content delivery and other functionality to help businesses scale and grow."

AWS is a global platform available in 190 countries. It provides its millions of customers to leverage its AWS cloud products and solutions to build sophisticated applications with increased flexibility, scalability and reliability. A user or customer can use and access AWS cloud computing services through AWS accounts; thus customers who own an account have access to cloud resources. A user with AWS account can create other users and grant them access to cloud resources inside this account and other accounts through federated identity and permissions. A user belongs to a unique account.

An access control model for AWS cloud services was developed by Zhang et al. [115]. This section briefly describes the AWS Access Control (AWSAC) model, which in turn forms a base

for the AWS IoT access control model developed in Chapter 4. The AWSAC model within a single AWS account is shown in Figure 2.9. As defined in [115], AWSAC has seven components: **Accounts (A), Users (U), Groups (G), Roles (R), Services (S), Object Types (OT),** and **Operations (OP)**. *Accounts* are basic resource containers in AWS, which allows customers to own specific cloud resources, and serve as the basic unit of resource usage and billing. *Users* represent individuals who can be authenticated by AWS and authorized to access cloud resources through an account. A user who owns an account can create other users inside that account and can assign them specific permissions on resources. *Groups* are a set of user groups. The *user_group* relation specifies the user to group assignment. If a policy is attached to a group, it will apply to all the users in that group, where users and groups belong to a single account.

"Roles" in AWS, unlike standard RBAC roles, are used for establishing trust relationships between users and resources in different AWS accounts. Users can be assigned roles through the *AssumeRole* action, and permissions assigned to these roles allows these users to gain access to corresponding cloud resources. The user-role mapping is specified through **virtual user_role** relation. The quotation marks are used for "roles" in Figure 2.9 to distinguish the AWS "roles" from RBAC roles. In the context of AWS, roles signify "roles" for simplicity. *Services* refer to AWS cloud services, such as compute, storage, networking, administration, database, etc. *Object Types* represents a specific type of object in a particular cloud service, such as virtual machines in the compute service EC2. *Operations* represent allowed operations on the object types based on an access control policy attached to them or their owning services.

AWS utilizes a policy-based access control mechanism. An AWS **Policy** is a JSON file which includes permissions defined on services and resources in the cloud. It comprises three main parts (or tags) *Effect, Action* and *Resources*, and optional *Conditions*. A policy can be attached to a user, a group, a role or a specific cloud resource. **Virtual Permission Assignment** is the process of virtually assigning permissions to users, roles, and groups by attaching policies to these entities. In cases where policy is attached to a resource, a specific *Principal* (an account, a user or a role) needs to be specified in the policy. There could be multiple permissions defined in one policy, and

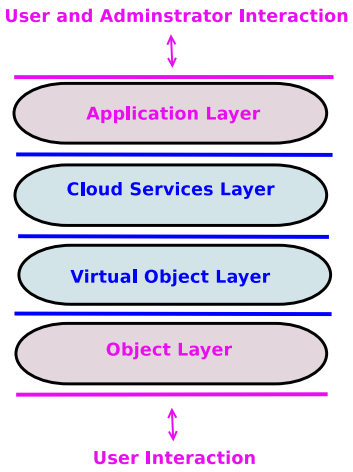


Figure 2.10: ACO Architecture for IoT [35]

multiple policies can be attached to one entity.

In Chapter 4, an abstract access control model for AWS IoT, known as AWS-IoTAC model, is developed by extending this AWSAC model. The AWS IoT platform is an instance of a CE-IoT architecture, more specifically a real-world CE-IoT platform provided by one of the largest Cloud services provider, Amazon Web Service (AWS) [1]. The principal goal of developing this model is to investigate and understand the authorization framework of a real-world CE-IoT platform by exploration of available documentation and hands-on-experiments. It is necessary to understand the current state-of-art of security in CE-IoT architecture and find what is lacking or needs improvement and propose appropriate solutions. Chapter 4 also discusses the need for an attribute-based approach to enhance IoT authorizations with fine-grained policies and proposes ABAC enhancement for the AWS-IoTAC model based on the realization of a Smart Home use case and its implementation in the AWS IoT platform.

2.5 ACO Architecture

Many layered IoT architectures, with variations in different layers of the architecture, have been proposed in the literature [33,37,45,49,57,72,82,85,90,91]. A general IoT architecture comprises three basic layers: *an object layer, one or more middle layers,* and *an application layer* [33,37,86, 110,111]. Recently, an IoT architecture, consistent with the above architectures, was proposed by

Alshehri and Sandhu in [35]. The motivation of their architecture is to integrate Cloud computing and its benefits in IoT and incorporate the concept of virtual objects (VOs) [82] which are the digital representation of physical objects. The authors have designed a layered IoT architecture with a focus on aiding the development of access control models for CE-IoT, and thus, named it as Access Control Oriented (ACO) architecture for CE-IoT.

Figure 2.10 shows the ACO architecture for IoT. It has four layers: *object layer, virtual object layer, cloud services layer,* and *applications layer*. Besides these layers, the ACO architecture also includes two other entities—*users* and *administrators*. Users are individuals who directly or indirectly interact with the IoT framework and benefit from its capabilities, while administrators are responsible for managing IoT securely and efficiently. The four layers of the ACO architecture are described as follows [35].

1. **Object Layer:** This is the base layer of the ACO architecture where heterogenous IoT devices like sensors, actuators, embedded devices, etc. reside. Mostly these devices are constrained devices with limited power, memory, and storage. Users directly interact with this layer while using and controlling the physical objects. For example, a user can manually turn off a device, such as a light or a pump. These devices or objects mainly collect data and send it to other endpoints, such as other objects, virtual objects, gateways, and cloud for storage, computation, and analysis. Object-to-object communication occurs within this layer and is achieved through various machine-to-machine (M2M) networking protocols and standards—Bluetooth, Zigbee, 6LoWPAN, ISA 100, WirelessHart/802.15.4, and LTE. The Internet enables the communication and data exchange with other ACO layers through a set of protocols, such as HTTP, MQTT, and CoAP [33, 35].

2. **Virtual Object Layer:** In the ACO architecture, the authors promote the use of virtual objects (VOs), that is the digital representation of physical IoT objects, and thus, introduce the virtual object layer to facilitate objects to applications interactions via VOs. VOs are capable of representing the current state of associated physical objects in the digital space when they are connected, and can also store a future state for those devices when they are

offline. They provide a uniform interface for the physical objects to communicate with the upper ACO layers. In AWS IoT [3], virtual objects are used to represent real-world IoT devices in the Cloud and are known as *Thing Shadows* or *Device Shadows*. Another capability of this layer is to enable VO-to-VO communication, irrespective of heterogeneous physical devices and their communication protocols. The authors in [35] also discussed different types of VO to physical object association—*one(or less)-to-one, many-to-one, one-to-many,* and *many-to-many* associations [82].

3. **Cloud Services Layer:** This layer is the core of the CEIoT architecture. Many researchers have presented the Cloud as one of the most important enabling technologies for IoT [33, 37, 45, 85, 90, 91]. The cloud services layer particularly serves to host the storage, computation, and analysis services for the huge amount of data generated by billions of IoT devices. These resource-constrained devices leverage the capabilities of the Cloud to perform desired functions. Users and business stakeholders can employ machine learning and data mining technologies to extract useful information from the IoT data that can be used in numerous ways to benefit customers.

   Besides this, cloud services layer is a suitable place to host an authentication and authorization service which manages secure communication and data access between IoT objects and applications. Different types of possible interactions, including communications and data access in this layer are: i) interactions between different cloud services inside one cloud (intra-cloud, cross-tenant, cross-account), ii) interactions between cloud services of different clouds (inter-clouds, multi-cloud), and iii) interactions between components of other layers (VO-to-Cloud, Cloud-to-Apps).

4. **Applications Layer:** This is the layer that delivers IoT services to end users through IoT applications and is the top-most layer of the ACO architecture. It acts as an interface for the users to remotely send commands and receive data and information from the objects. Users are also able to visualize the IoT data analyzed in the cloud services layer through the

applications. The applications also allow administrators and users to configure devices and define access control polices for securing access to IoT resources and data.

This four-layer architecture focuses on an overall general IoT space. In some IoT application domains, such as Wearable Internet of Things (WIoT) [62], where devices are small in size, heterogeneous in nature (e.g., networking protocols, standards, vendors), and are highly resource constrained. Hence, an abstraction layer that abstracts the heterogeneity and enables communications between these devices and other components in the layered architecture is necessary. Also, within each layer and among different layers of the ACO architecture, the access and communication control requirements need to be addressed through appropriate access and communication control models.

An enhanced ACO (EACO) architecture with five layers and an Access control framework based on different types of communications in the EACO architecture is presented in Chapter 5. A novel attribute-based approach for securing communications, i.e., Attribute-Based Communication Control (ABCC) is also introduced along with an abstract ABCC model for CE-IoT in Chapter 6.

CHAPTER 3: ABAC MODELS AND ENFORCEMENT FOR CLOUD IAAS UTILIZING THE POLICY MACHINE

This chapter presents two attribute-based access control models developed in the context of Cloud Computing, viz. the User-Attribute Enhanced OSAC (UAE-OSAC) model for the OpenStack IaaS platform, and the restricted HGABAC (*rHGABAC*) model. For enforcing these models in real-world platforms, this dissertation utilizes a novel enforcement architecture consisting of the Policy Machine (PM), a reference implementation for the Next Generation Access Control (NGAC), ANSI/INCITS standard by NIST, and the Authorization Engine (AE), a RESTful service developed in this research as an interface to the PM. Significant portions of this chapter build on the following publications [38, 39] with some revisions and updates.

- Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. An Atribute-Based Access Control Extension for OpenStack and its Enforcement Utilizing the Policy Machine. In *2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 37-45. IEEE, 2016.

- Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 17-28. ACM, 2017.

3.1 User-Attribute Enhanced OSAC (UAE-OSAC) Model

In industry, RBAC [54, 92, 93] has been the dominant access control model since the 1990s. It is a widely accepted access control mechanism for many applications. For example, in addition to OpenStack [100], other major cloud services providers like AWS [1], and Microsoft Azure [21] utilize some customized form of RBAC for their authorization architecture.

Besides many well-known advantages of RBAC [55], it also has some well-known limitations [89], such as role explosion and role-permission explosion. Due to the limitations of RBAC, there is a shift towards Attribute-Based Access Control (ABAC) models to enhance flexibility and

achieve fine-grained access control by using attributes, beyond roles and groups, of involved entities in the authorization process. This shift has to be gradual since it is unrealistic for existing systems to abruptly adopt ABAC models, completely eliminating current RBAC implementations. Therefore, an authorization mechanism combining both RBAC and ABAC should be considered for such platforms, which later can be adapted to be purely ABAC.

NIST has identified three different ways of combining RBAC and ABAC effectively by adding attributes to role-based access control policies as follows [69, 74].

- *Dynamic Roles*: This approach uses user attributes and context attributes to assign specific roles to the users dynamically. One of the examples of this approach is attribute-based user-role assignment [34].

- *Attribute-Centric*: Here, the user roles are just another attribute of the users with no special semantics. That is, a role does not have any special significance, such as permissions assigned to it or hierarchical relationship with other roles.

- *Role-Centric*: In this method, user permissions gained through roles are further constrained based on user attributes and other relevant attributes (e.g., environment attributes).

Using just the user roles for assigning permissions on Cloud resources is a restrictive authorization framework, which will soon lead to role-explosion problem in the Cloud IaaS platform. To secure access to data and resources in the Cloud, more flexible access control approach is required. Therefore, this research proposes and develops a role-centric ABAC model for a Cloud IaaS platform, viz. OpenStack, known as the User-Attribute Enhanced OpenStack Access Control (UAE-OSAC) model. It is developed by extending the existing OpenStack Access Control (OSAC) model.

3.1.1 UAE-OSAC: Motivation

OpenStack is a leading open source cloud services platform with a rigorous development and release cycle of every six months. With its huge customer base and services, OpenStack will eventu-

ally or may already be facing the role explosion or role-permission explosion problem. Therefore, it is inevitable for OpenStack to adopt a more flexible access control mechanism, such as ABAC. However, with so many previous releases and continuous ongoing development, it is difficult to fully replace the RBAC authorization foundation by ABAC.

In RBAC, access control policies can only be defined on the basis of roles which restricts access control flexibility. Whereas, ABAC provides great flexibility to express fine-grained access control policies in a simple and more powerful way based on attributes of users, subjects, and objects [63, 64, 67]. With the advancements of ABAC and its capabilities, it is essential to develop ABAC models for real-world applications and systems. However, it is difficult for existing systems to instantaneously adapt to attribute-based access control policies since they require a well-defined and robust attribute and access control management system for their implementation. The transformation from RBAC to ABAC policies has to be gradual but we will eventually see wide adoption and implementation of ABAC models in the industry. In particular, combining ABAC with roles is one promising transition path as discussed by NIST [74].

The UAE-OSAC model is a first step towards addressing security threats from unauthorized users and attackers to the OpenStack Cloud IaaS platform. The fundamental motivation for the user-attribute enhanced OSAC model is to enhance current RBAC authorization framework of OpenStack with the features of ABAC to combine the best of both models. This provides the benefits of RBAC together with enhancing access control flexibility with support of user attributes, while minimizing the overhead of altering the existing OpenStack access control framework.

3.1.2 UAE-OSAC: Model and Definitions

This subsection discusses and formally defines the User-Attribute Enhanced OSAC (UAE-OSAC) model. UAE-OSAC is a role-centric ABAC model for OpenStack which adds user attributes to the core OpenStack Access Control (OSAC) model developed in [105]. It provides a fine-grained access control mechanism where roles determine the initial set of permissions for the users which are further refined by ABAC policies defined based on user attributes.

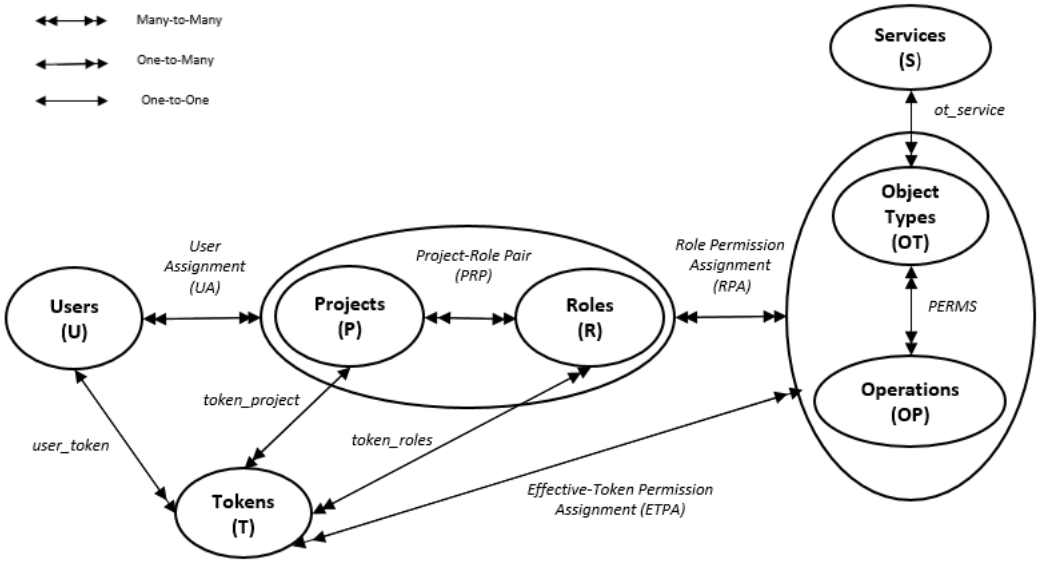


Figure 3.1: Simplified OpenStack Access Control (OSAC) Model (Adapted from [105])

For developing the UAE-OSAC model, the core OSAC model [105] has been simplified by removing *Groups* and *Domains*. The simplified OSAC model is shown in Figure 3.1 and is formally defined in Table 3.1. It comprises seven entities: **users, projects, roles, services, object types, operations,** and **tokens**. **Groups** and **domains** are removed in this simplified model since groups are collection of users, and users in OpenStack belongs to a single domain or tenant (where they are created) and can be seen and managed only by the domain owner or administrator. The scope of simplified OSAC is within a single domain. Thus the administrative boundary of domains is not relevant in this context.

The core components of the OSAC model are previously described in Chapter 2. Thus a brief description is presented here. The focus is on new derived model components and the UAE-OSAC model components. **U, P, R, S, OT, OP,** and **T** are finite sets of users, projects, roles, services, object types, operations, and tokens respectively. There is a set of project-role pairs **(PRP)** which is assigned to the users and is represented through user and project-role assignment defined by **UA**. Similarly, **RPA** represents the assignment of permissions to the roles. The **ot_service** function maps the object types to specific services in the Cloud.

When the user authenticates to OpenStack, she receives a **token** from the identity service,

Table 3.1: Simplified OSAC Model and its Core and Derived Components (Adapted from [105])

Definitions 1.
1.1 Core Components
• U, P, R, S, OT, OP and T are finite sets of users, projects, roles, services, object types, operations and tokens respectively • $PRP = P \times R$, is the set of project-role pairs • $PERMS = OT \times OP$, is the set of permissions • $UA \subseteq U \times PRP$, is a many-to-many user to project-role assignment relation • $RPA \subseteq PERMS \times R$, is a many-to-many permission to role assignment relation • $ot\_service : OT \rightarrow S$, is a function mapping an object type to its associated service • $user\_tokens : U \rightarrow 2^T$, is a function mapping a user to a set of tokens; correspondingly, $token\_user : T \rightarrow U$, is a mapping of a token to its owning user
1.2 Derived Components
• $token\_roles : T \rightarrow 2^R$, is a function mapping a token to its set of roles, formally, $token\_roles(t) = \{r \in R \mid (token\_user(t), (token\_project(t), r)) \in UA\}$ • $ETPA : T \rightarrow 2^{PERMS}$, is a function specifying the permissions available to a user through a token, formally, $ETPA(t) = \bigcup_{r \in token\_roles(t)} \{perm \in PERMS \mid (perms, r) \in RPA\}$.

viz. Keystone. This token represents the scope of user's accesses on the Cloud resources. The **user_tokens** is a mapping between users and their tokens. The derived components are derived from the core components which are **token_roles** and **Effective Token Permission Assignment (ETPA)**. *ETPA* is based on the token a user presents during access requests, and permissions are identified based on roles present in a token for a specific user, in specific projects.

Figure 3.2 presents the User-Attribute Enhanced OSAC (UAE-OSAC) model. It has all the core components and derived components of simplified OSAC model along with newly added entities and relationships. **User attributes** are added to the *Users* and a new relationship **UAPA** is introduced for user-attribute value and permission assignment. This model is a role-centric ABAC [74] model in the sense that it incorporates the existing RBAC framework of OpenStack, keeping all its advantages, and adds in the flexibility of ABAC model by introducing user attributes. A user's roles determine the maximum permissions which are further reduced or constrained by user attribute permission assignments. Table 3.2 presents definitions for the newly added components.

User Attribute is a function which takes a user and returns a specific value from its range, where the **range** of an attribute is a finite set of atomic values defined for each attribute function.

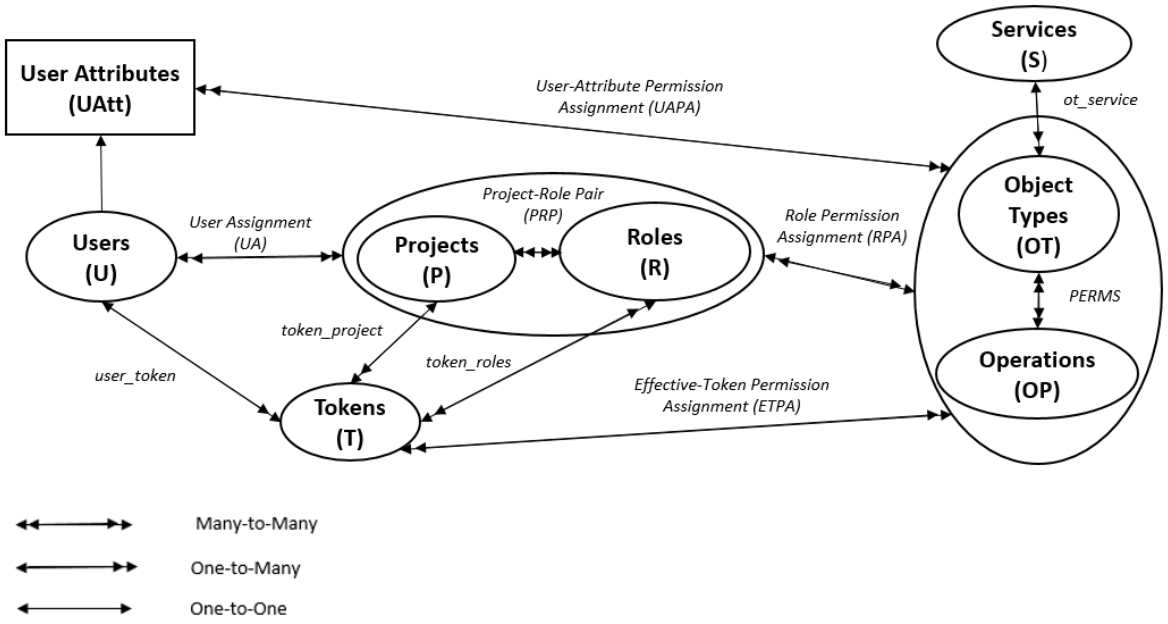


Figure 3.2: User-Attribute Enhanced OSAC in Single Tenant

Table 3.2: UAE-OSAC Model and its Components

Definitions 2.
2.1 Additional Core Components
• $UAtt$ is a finite set of user attribute functions • $Range(uatt)$ where $uatt \in UAtt$, is a finite set of atomic values defined for each user attribute function in $UAtt$ • For each $uatt$ in $UAtt$, $uatt : U \rightarrow Range(uatt)$, is a mapping of each user to an atomic value in $Range(uatt)$ • $UAPA \subseteq PERMS \times \{\langle uatt, v\rangle \mid uatt \in UATT; v \in Range(uatt)\}$, is a many to many permission to attribute-value assignment relation
2.2. Modified Derived Components
• $ETPA : T \rightarrow 2^{PERMS}$, is a function specifying the permissions available to a user through a token and user-attribute value assignment, formally $ETPA(t) = \bigcup_{r \in token\_roles(t)}\{perm \in PERMS \mid (perm, r) \in RPA\} \cap \bigcup_{uatt \in UAtt}\{perm \in PERMS \mid (perm, \langle uatt, uatt(token\_user(t))\rangle) \in UAPA\}$

In general, attributes are of two types: *atomic-valued* and *set-valued*. An atomic-valued attribute returns only one value from its range, whereas a set valued attribute returns a subset of values from the range of the attribute. The attributes of a user represent its characteristics and properties. Some examples of user attributes are Department, Clearance, and Specialization [67, 69].

The user attributes in the UAE-OSAC model are atomic-valued attributes. **UAPA** represents a new relationship between user attributes and permissions. It is a set of permissions associated with the user attributes and their assigned values. Consequently, the **ETPA** relationship has been modified to include permissions from user attributes, in addition to permissions included from user roles. One of the main objectives of this model is to depict applications of ABAC model in real-world Cloud IaaS applications. Therefore, the UAE-OSAC model needs to be enforced in the OpenStack platform, and its enforcement details are discussed in the following section.

3.1.3 Enforcement Utilizing the Policy Machine and Authorization Engine

While theoretical models are essential to enriching the state-of-art of ABAC, practical implementations for enforcing these models in real-world scenarios are equally crucial for their wide adoption and applicability in the industry. This section presents the enforcement and implementation details of applying the UAE-OSAC model in OpenStack utilizing the Policy Machine (PM) and a proof-of-concept implementation, the Authorization Engine (AE). PM is a reference implementation of the Next Generation Access Control (NGAC) standard proposed by the National Institute of Standards and Technology (NIST). It is an open-source freely available software and can be modified as per the requirements. PM enables expression and enforcement of different types of access control policies, for example, DAC, MAC, and RBAC, through its policy configuration points. It provides a unifying framework to define, administer and enforce commonly known access control policies as well as new access control policies. A detailed description of the PM has been presented in Chapter 2.

Similarly, OpenStack is a rapidly changing open-source cloud platform that provides an architecture to use or enhance its services as per the users or customers requirements. OpenStack and

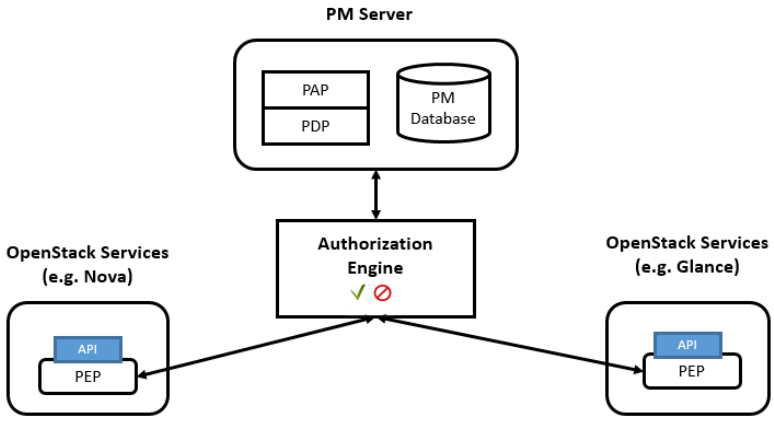


Figure 3.3: An ABAC Enforcement Architecture for OpenStack using PM

PM, both being open-source, offer the capability to integrate a custom authorization component implementation in the OpenStack authorization framework. A customized authorization engine is necessary to enforce the UAE-OSAC model on the OpenStack platform using the PM. Thus, an authorization engine (AE), which is a RESTful service, has been implemented as a proof-of-concept which acts as an interface between OpenStack and PM. In the enforcement framework, OpenStack Kilo release with Identity API version 2 and PM version 1.5 (Harmonia 1.5) have been used. A newer version of the PM, Harmonia 1.6, is released recently with new features and better performance. However, this research is based on Harmonia 1.5 since the newer version released after the research has been already conducted. It would be interesting to explore Harmonia 1.6 features and capabilities in the future work.

PM is a flexible attribute-based access control framework which allows to specify, enforce, and combine different types of access control policies. Therefore, PM with its flexible approach and capabilities is a suitable choice to build the enforcement framework for the UAE-OSAC model.

A. Enforcement Architecture

This dissertation introduces a novel enforcement architecture for enforcing ABAC policies in Cloud IaaS platforms utilizing the Policy Machine (PM) [25, 51, 52], an attribute-based policy specification and enforcement tool developed by NIST, and an Authorization Engine (AE), a proof-

Figure 3.4: OpenStack Policy in PM

of-concept implementation. Figure 3.3 shows the enforcement architecture. In this architecture, the PM server acts as a centralized policy administration point that returns a set of user permissions on objects based on the policy definitions. It connects to an active directory (AD), a back-end database for PM that stores all the users and their associated user attributes. For simplicity, OpenStack is assumed to be using the same AD as its user identity back-end to store all users related information, including attributes.

The PM based enforcement architecture utilizes a server-client architecture where PM server and AE has a server-client relationship, and similarly AE, itself, acts as a server for different services of OpenStack. OpenStack services communicate through a RESTful API to AE which in turn communicates to PM server for making authorization decisions (e.g., Allow/Deny). Due to dynamic nature of cloud objects, the commands in OpenStack are modeled as objects in the authorization policy defined in PM. These commands are specific to services in OpenStack, for example, Nova has its own commands, and similarly other services, such as Glance, Cinder, etc. The required authorization policy definitions, typically listed in OpenStack policy file (a JSON document), have been defined in PM Admin Tool in a policy class named, *OpenStack*.

An instance of OpenStack policy class, from "Objects/Attributes with ACE's" view in the PM tool, is shown in Figure 3.4. The figure shows a sample access control policy defined for Nova *keypair* commands in OpenStack. These commands generate ssh keys for a user which in turn are

used while creating VMs in Nova. In Figure 3.4, *Admin* and *Manager* are roles (depicted as user containers/attributes in PM), and *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show* are Nova commands (depicted as objects in PM). These user attributes and objects are associated with a specific operation set randomly named, e.g., *"06932852"* is an operation set with "read" permissions only between user attributes—*Admin* and *Manager*, and object—*compute_extension-keypair-index*. It means that the PM attributes *Admin* and *Manager* have *read* permissions on *compute_extension-keypair-index*, so that a user with any of these roles is authorized to do this operation in OpenStack.

B. Authorization Engine

This section discusses the Authorization Engine (AE), a proof-of-concept implementation for enforcing the UAE-OSAC model in OpenStack utilizing the PM. Among many advantages of the PM, it is an open source tool which led to the development of the Authorization Engine (AE). AE acts as an authorization component facilitating communication between the OpenStack policy engine and the PM. It is a RESTful service that provides an interface between OpenStack and the PM by getting attribute information and permission list from the PM server and evaluating authorization decisions. The authorization decision, either *Allow* or *Deny*, is then returned to the specific OpenStack service where the policy is enforced.

It is written in Java using a REST API and acts as a RESTful server for OpenStack services. For any operation to be performed by a *user* in OpenStack, AE initially verifies the *project* in the target and the *token*. Then, it connects to the PM server and queries it via different PM commands to make access control decisions based on the UAE-OSAC model for OpenStack.

AE replaces the existing policy engine in OpenStack and is responsible for evaluating the policy defined in PM and returning access decisions to OpenStack services. OpenStack Services (S) are the policy enforcement points (PEP) that enforce the access decisions returned by AE and responds to the users with appropriate results, such as provide requested information if access allowed, or return an error if access denied.

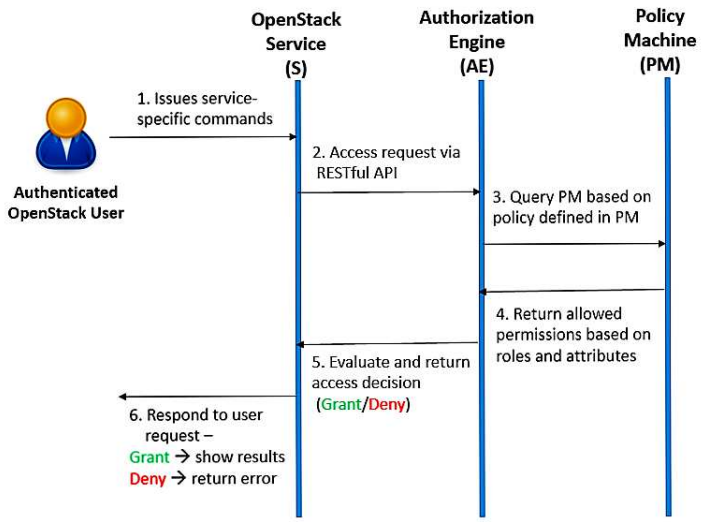


Figure 3.5: OpenStack Authorization using AE and PM

Figure 3.5 depicts a sequence diagram presenting authorization in OpenStack using AE and PM. It shows the sequence of actions involved in the authorization process. Since AE is a proof-of-concept implementation and is mainly designed to depict the applicability and feasibility of the proposed UAE-OSAC model in OpenStack Cloud IaaS platform, it has not been optimized for performance. AE can be designed to be a more general and independent component which could be used with any policy-configuration tool, like PM, and can be applicable to other cloud platforms besides OpenStack.

C. Use Cases: RBAC and Role-Centric ABAC Policies

This section presents use cases with two types of access control policies, first with only user roles, and second with user roles and user attributes. These use cases illustrate how existing RBAC and proposed role-centric ABAC policy would work in an organization-specific environment in OpenStack. They are configured in simplified OSAC model and a user-attribute enhanced OSAC model respectively and can be enforced in OpenStack using the PM and AE. They depict the added benefits of the UAE-OSAC model.

For the UAE-OSAC model, AE has been customized to work with two types of policies, an RBAC policy (same as OpenStack's current access control policy), and a role-centric ABAC policy

with user attributes (the user-attribute enhanced OSAC policy). However, it can be easily extended to enforce other types of access control policies as required.

- **A Simplified OSAC RBAC Policy:**

The first use case presents a RBAC policy, equivalent to simplified OSAC policy in OpenStack, with two roles: *Admin* and *Manager*, and four *Nova_commands*: *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*. Similarly, other commands in different services of OpenStack can be incorporated in the authorization policies. The user permissions are determined by the roles that a user is assigned in a specific project. The Nova *keypair* commands are used to generate ssh keys for a user. These keys are used while creating VMs. A user can create, delete, list, and show details of keypairs using these commands. The authorization rules for each command, for a generic user *u* are given below.

Roles: *{Admin, Manager}*

Commands (c): *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*

Authorization rules for any user u:

- *compute_extension-keypair-create* $\rightarrow$ *Role(u) = Admin*
- *compute_extension-keypair-delete* $\rightarrow$ *Role(u) = Admin*
- *compute_extension-keypair-index* $\rightarrow$ *(Role(u) = Admin* $\vee$ *Role(u) = Manager)*
- *compute_extension-keypair-show* $\rightarrow$ *(Role(u) = Admin* $\vee$ *Role(u) = Manager)*

Here, the rules state that a user must have an *Admin* role to create or delete keypairs, whereas *compute_extension-keypair-index* and *compute_extension-keypair-show* are authorized to be performed by an *Admin* role or a *Manager* role. To enforce this authorization policy, an equivalent authorization policy is specified in the PM. Figure 3.6 shows the policy defined in PM. There

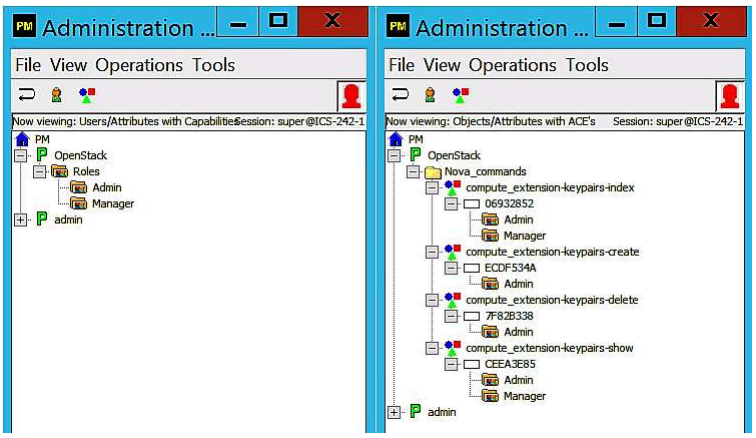


Figure 3.6: A Role-Based Access Control Policy in PM

```
stack@opm-1:/opt/stack/nova/nova$
stack@opm-1:/opt/stack/nova/nova$ nova --os-username user1 --os-password ****** --os-tenant-name test
keypair-add test4 >test4.pem
stack@opm-1:/opt/stack/nova/nova$ nova --os-username user1 --os-password ****** --os-tenant-name test
keypair-list
+-------+-------------------------------------------------+
| Name  | Fingerprint                                     |
+-------+-------------------------------------------------+
| test  | ************************************************* |
| test1 | ************************************************* |
| test2 | ************************************************* |
| test3 | ************************************************* |
| test4 | ************************************************* |
+-------+-------------------------------------------------+
stack@opm-1:/opt/stack/nova/nova$ nova --os-username user2 --os-password ****** --os-tenant-name test
keypair-add test5 >test5.pem
ERROR (Forbidden):  Policy doesn't allow [compute_extension:keypairs:create] to be performed for role
[Manager] due to role (HTTP 403) (Request-ID: req-88a0af9a-d6ae-46d5-b308-3b59a2fa2908)
stack@opm-1:/opt/stack/nova/nova$
```

Figure 3.7: OpenStack Enforcement Results

are two roles defined *Admin* and *Manager* as shown on the left side, and associations between commands and roles via operation sets (alpha-numerically named by default) are shown on the right in Figure 3.6. As shown on the right side, *compute_extension-keypair-index* can be done by *Admin* or *Manager*, whereas only Admin can perform *compute_extension-keypair-create* and *compute_extension-keypair-delete*. Similarly, at the OpenStack end, two roles *Admin* and *Manager* are defined. Either one or both of these roles are assigned to some users in a test tenant, for example *user1* is an *Admin* and *user2* is a *Manager*. The policy was tested in OpenStack by executing different commands for these users. A screenshot of the authorization results for few commands is shown in Figure 3.7.

- **A Role-Centric ABAC Policy:**

This subsection presents a role-centric ABAC policy with user attributes added to the above

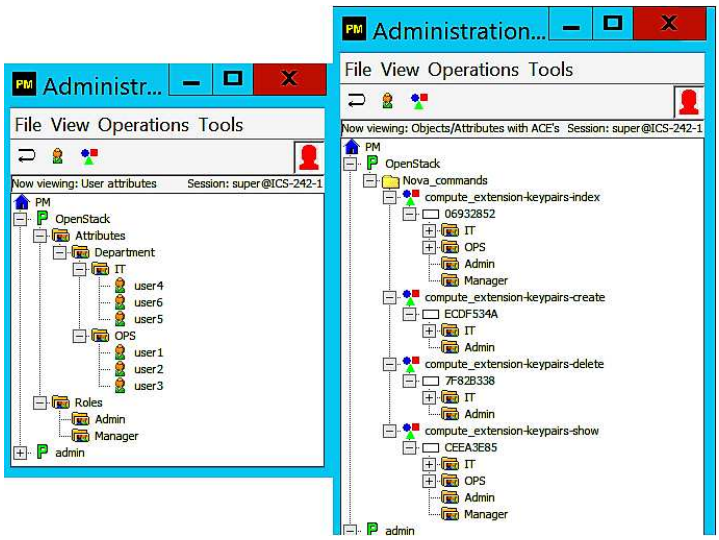


Figure 3.8: A User-Attribute Enhanced OSAC Policy in PM

policy. This is an example of the user-attribute enhanced OSAC policy. Besides roles and commands, there is a user attribute—*Department* that can be assigned only one value from its *Range* = $\{IT, OPS\}$ for any user. Here, the user attribute is atomic valued unlike roles, which implies that a user can have multiple roles assigned, but it can only be in one department, either *IT* or *OPS*. For any user, accesses are defined based on their roles and their associated user attributes. In a real organization, there are multiple roles and user attributes, and permissions are assigned based on roles and user attributes.

The authorization policy is defined based on roles and user attributes and is given below. As per the first rule, a user u is authorized to *compute_extension-keypair-create* only if the user is assigned role *Admin* and the user is in department *IT*. The logical AND ($\wedge$) symbol implies that both of the conditions must be true. Otherwise, a user without the specified role and user-attribute value will be denied access. The second rule is similar. In the third rule, a user u needs to have an *Admin* or *Manager* role along with user-attribute value *IT* or *OPS*. This is a role-centric policy which means that first the role is checked, then the user-attribute value will be checked. If a role check fails, then access is denied, and user-attribute value is not checked.

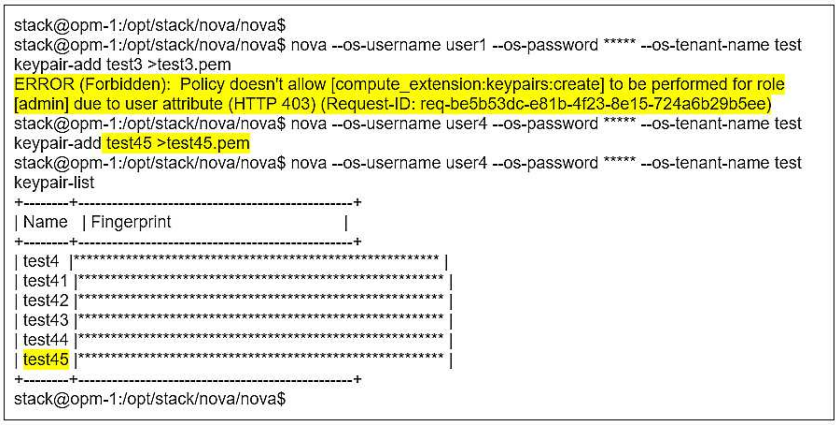


Figure 3.9: OpenStack Enforcement Results

Roles: *{Admin, Manager}*

Department: *{IT, OPS}*

Commands (c): *compute_extension-keypair-index, compute_extension-keypair-create, compute_extension-keypair-delete,* and *compute_extension-keypair-show*

Authorization rules for any user u:

- $compute\_extension\text{-}keypair\text{-}create \rightarrow (Role(u) = Admin \wedge Dep(u) = IT)$

- $compute\_extension\text{-}keypair\text{-}delete \rightarrow (Role(u) = Admin \wedge Dep(u) = IT)$

- $compute\_extension\text{-}keypair\text{-}index \rightarrow ((Role(u) = Admin \vee Role(u) = Manager) \wedge (Dep(u) = IT \vee Dep(u) = OPS))$

- $compute\_extension\text{-}keypair\text{-}show \rightarrow ((Role(u) = Admin \vee Role(u) = Manager) \wedge (Dep(u) = IT \vee Dep(u) = OPS))$

Figure 3.8 demonstrates the specification of this policy in the PM. Unlike the previous use case, there is an additional PM container *Attributes* which contains user attribute *Department* with two values *IT* and *OPS*. The associations are shown on the right side and include both the user attribute and the roles.

On the OpenStack side, few users with different roles and user attribute values are defined, and accesses for them is tested against the desired results. For example, a user *user1* is defined

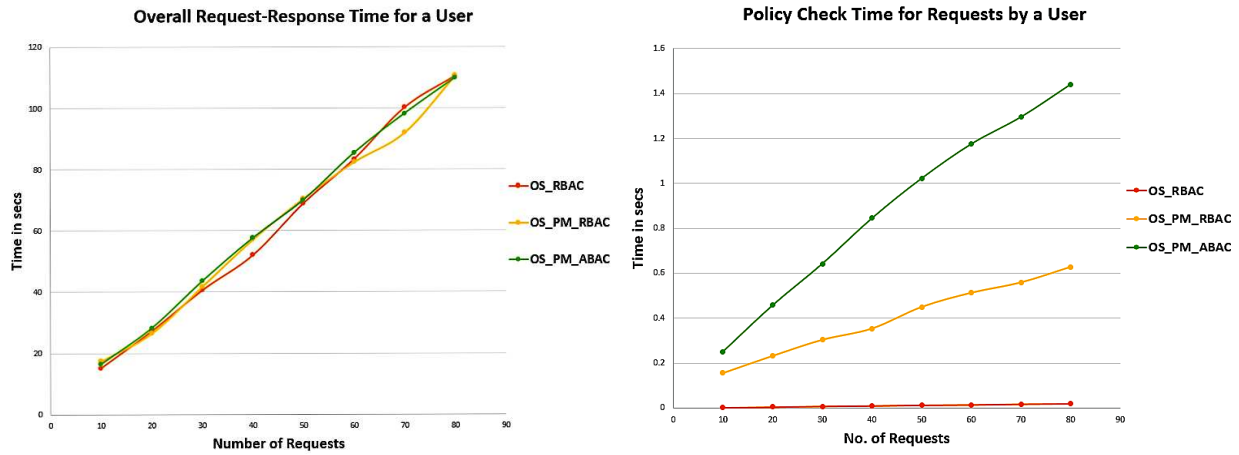
with role *Admin* and user attribute department as *OPS*, and another user *user4* is defined with role *Admin* and user attribute department as *IT*. In this case, *user4* has access to execute command *compute_extension-keypair-create*, whereas *user1* is denied access due to the department attribute value *OPS*. Figure 3.9 presents a screenshot of the results in OpenStack.

D. Performance Evaluation

This section analyzes and discusses the performance evaluations of the ABAC extension with user attributes to OpenStack employing the above policy use cases and their enforcement utilizing the PM and AE based enforcement architecture. Generally, there is always some trade-off between performance and enhanced functionality or capability. Here, the main goal is to demonstrate the feasibility of the UAE-OSAC model in the OpenStack platform. The UAE-OSAC model incorporates all the benefits of existing OpenStack access control mechanism and adds great flexibility by adding user attributes. These user attributes included in the access control policy allows a user to be assigned least possible permissions and allows to define more fine-grained access control policies avoiding problems such as role explosion and role-permission explosion.

Generally, an admin user is supposed to have maximum permissions but in a real enterprise, there are various departments, and there could be an admin for each department. Similarly, many employees are working in a department having different skill levels. Thus, creating a role for each of these combinations result in role explosion. However, having user attributes such as department and skills along with role admin in a policy avoids such problems and significantly enhances the capability and flexibility of access control policy.

For conducting the performance evaluation, the experiments were performed using two types of policies discussed in the use cases. We created scripts to evaluate the overall request-response time and the policy evaluation time for each Nova command executed by a specific user. A number of requests (Nova commands) were executed for different users. The graph in Figure 3.10(a) shows the overall request-response time for a set of requests run by an OpenStack user. Whereas, Figure 3.10(b) depicts mainly the policy evaluation time taken to evaluate the access control policies and



(a) Overall Time Taken in Requests-Response

(b) Policy Check Time for Requests by User

Figure 3.10: Performance Evaluation for UAE-OSAC Model

get an access decision when an access request was made.

The graphs in Figure 3.10 contain three curves, one for a RBAC policy in OpenStack without any modifications (*OS_RBAC*), second for the same RBAC policy in OpenStack with modified policy engine, i.e., AE and the PM (*OS_PM_RBAC*), and third for the user-attribute enhanced role-centric policy in OpenStack with AE and PM (*OS_PM_ABAC*). The overall request-response time for these three cases is quite similar. However, the policy evaluation time for each of these cases differs from each other. A centralized policy administration point, the PM, and a RESTful service AE in between PM and OpenStack adds latency in the policy evaluation time. Some of few techniques identified to enhance performance are: i) using performance enhanced server to host PM and AE in order to improve policy evaluation time, ii) caching policy evaluation results locally on the OpenStack services, and iii) having PM, AE, and OpenStack services installed on an isolated subnet, similar to a typical OpenStack installation in a production environment to reduce communication latency in the network.

Performance evaluation of the UAE-OSAC model gives an idea of the cost of applying it in real-world systems. A number of reasons could have contributed to the performance latency. First, the enforcement architecture is a proof-of-concept implementation and hasn't been optimized for performance specifically. It has been instead driven by the objective to depict applicability and

feasibility of the model in a real platform, i.e., OpenStack. PM, the tool utilized to enforce this model, is mainly designed as a general attribute-based access control framework but needs some performance enhancements to be used in a real production environment. Also, there is network latency contributing to the time for each request from OpenStack to a centralized PM server which is currently residing on a node in a different subnet.

3.2 Restricted HGABAC (*rHGABAC*) Model

An ABAC model with user groups and object groups with the hierarchical relationship among these groups is proposed by Servos and Osborn which is known as the Hierarchical Group and Attribute-Based Access Control (HGABAC) model [101]. Gupta and Sandhu presented a conceptual model of the HGABAC model with an alternate formalization in [58] for developing an administrative access control model for HGABAC. It is a logical-formula authorization policy model. A review of this conceptual HGABAC model along with its formal definitions has been presented in Chapter 2. A role-centric ABAC extension for OpenStack is proposed in the previous section along with its enforcement utilizing the PM and AE. Whereas, this section presents a pure ABAC model, adapted from the HGABAC model, named the restricted HGABAC (*rHGABAC*) model.

In contrast to the HGABAC model, the *rHGABAC* model is developed here as an enumerated authorization policy (EAP) model, more specifically a single-value EAP. In a single-value EAP, a policy tuple comprises of a single value of user attribute and a single value of object attribute. Due to this property, the *rHGABAC* model is restricted in nature and is not capable of expressing some kinds of policies which can be represented in HGABAC or other types of enumerated policies. An example of a policy that can't be expressed in *rHGABAC* is where a user who is both a *Manager* and an *Admin* can access objects of type *Private*. A different policy where a user who is a *Manager* or an *Admin* can access objects of type *Private* can be easily expressed in *rHGABAC*. In other words, *rHGABAC* cannot express conjunctive policies.

In addition to HGABAC capabilities (e.g., groups and group hierarchy), this dissertation introduces attribute hierarchy, a partial order relationship between attribute values in the range of

attributes, in the *rHGABAC* model. Different types of attribute hierarchies have been discussed in the literature [43, 76]. To implement this model in a real-world scenario, this section utilizes the enforcement architecture with the PM and AE, presented in the previous section. However, the prior architecture is focused on the OpenStack platform. This section develops a more generalized enforcement architecture so that the *rHGABAC* model can be implemented in any application or platform (with some required modifications) that supports RESTful communications, for example, the OpenStack IaaS Cloud platform or any other RESTful application platform.

PM allows to define attribute-based access control policies; however, the attributes in PM are different than attributes in typical ABAC models as name-value pairs. The attributes (user and object attributes) in PM are containers, for example, PM's user attributes are containers that contains a set of users. Suppose a role *Manager* is represented as a user attribute in PM, then it will contain all the users (e.g., Alice, Bob, etc.) who have the role *Manager* assigned to them through the *assignment* relation. Therefore, to overcome this disparity between ABAC attributes and PM attributes, this research identifies a suitable policy configuration mechanism for the proposed *rHGABAC* model employing PM capabilities. It also demonstrates use cases, their configuration in PM, and implementation using a generalized authorization architecture.

3.2.1 *rHGABAC*: Motivation

Recently, ABAC research has gained momentum with a surge of interest from different sectors developing in ABAC models. It is mainly inspired by the limitations of traditional access control models, viz. DAC, MAC, and RBAC. Numerous ABAC models [38, 65, 67, 87, 103, 112] have been proposed. Despite the existence of these different ABAC models, there is no consensus on a specific standard ABAC model. However, a well-accepted simplest form of attribute-based access control includes users, user attributes, objects, object attributes, actions, and permissions or operations allowed for users on specific objects, based on attributes of users and objects. This typical foundation provides great freedom to researchers to incrementally enhance this basic ABAC architecture based on customized needs and requirements in different scenarios. Any additional

component that uses or is compatible with the basic ABAC components can be incorporated with them to get a more powerful and flexible ABAC model.

Besides basic ABAC models, there are models designed with additional capabilities such as groups, and group attributes and hierarchies [43, 101]. However, implementation and demonstration of such ABAC models in real-world applications are still lacking. This dissertation develops an EAP hierarchical ABAC model (*rHGABAC*) with groups, group attributes, and group and attribute hierarchy which is built upon a conceptual HGABAC model presented in [58]. Hierarchical relationship among groups and attributes enhances access control flexibility and facilitates attribute management and administration. It allows to define more fine-grained access control policies. Moreover, EAP ABAC allows to simplify the management and administration of authorization policies by simplifying the policy review and policy update tasks.

Although ABAC research has received significant attention in academia, it is not so common to find implementations of these models in the industry. There are a few existing tools such as XAMCL [32] and Policy Machine (PM) [25, 51, 52] that are capable of expressing different types of attribute-based access control policies. However, wide adoption of these tools remains a challenge. A general authorization architecture employing the PM and custom built AE is proposed in this research to fill the gap between the existence of these tools and their adoption in real-world scenarios.

Consistent with the *rHGABAC* model, PM is also a single-value enumerated authorization policy which makes it a feasible choice for implementing the *rHGABAC* model. Although PM hasn't been specially designed for group attributes and hierarchical relationships, these features could be realized in PM through its containment property. The containment property, as defined earlier, exists among PM attributes through its *assignment* relation. It can be defined as: if there exist any two elements x and y such that x is assigned to (contained in) y by one or more assignment relations, then x acquires or gets all the properties and capabilities of y in addition to its own directly conferred properties [52].

The fundamental motivation behind the *rHGABAC* model is to include features, such as groups,

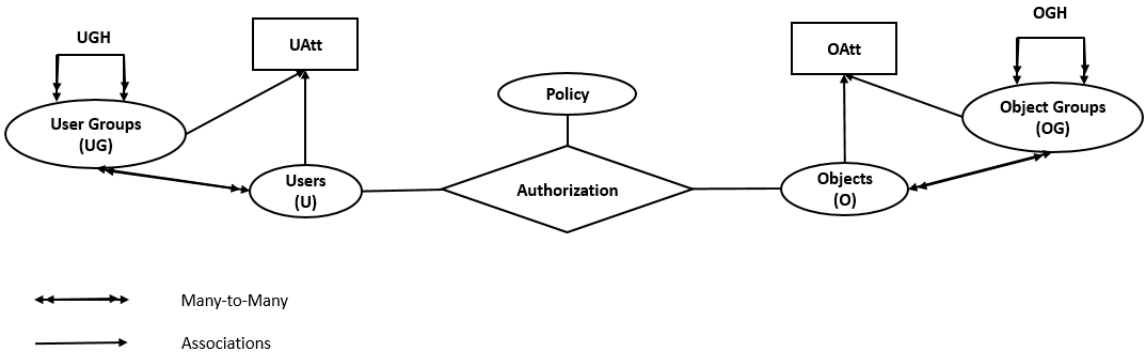


Figure 3.11: The *rHGABAC* Model (Adapted from [58, 101])

groups attributes and hierarchy, and attribute hierarchy, i.e., a hierarchical relationship among attribute values, in an ABAC model and show its implementation as enumerated policy utilizing an enumerated policy tool, the PM. Note that HGABAC requires attributes to be set-valued so inheritance can be seamlessly achieved. Atomic-valued attributes, which can have only one value, would require some conflict resolution in the presence of inheritance. Therefore, this research follows the HGABAC approach.

3.2.2 *rHGABAC*: Model and Definitions

Servos and Osborn [101] proposed a hierarchical ABAC model known as the hierarchical group and attribute-based access control (HGABAC) model. In addition to basic ABAC components, this model includes user and object groups and introduces attribute inheritance through hierarchy among groups. It also includes environment, connection, and administrative attributes in access control policies. A group-based hierarchical assignment of user and object attributes is a novel part of this model.

As discussed earlier, HGABAC is a logical-formula based authorization policy model. However, this section presents a restricted HGABAC model, named *rHGABAC*, and formalize it as a single-value EAP. In the process of developing an administrative model for HGABAC, named GURA_G, Gupta and Sandhu presented a conceptual model of HGABAC with an alternate formal-

ization for it based on $ABAC_\alpha$ [58]. The *rHGABAC* model has been adapted from their model; however, it is formalized as a single-value EAP policy. The model is shown in Figure 3.11 with its formalization presented in Table 3.3.

The main components of this model are **users, user groups, user attributes, objects, object groups, object attributes, operations, policy,** and **authorization**. There is a hierarchical relationship (i.e., a partial order) among groups—user group hierarchy **(UGH)** and object group hierarchy **(OGH)** through which attribute inheritance is achieved. For simplicity, subjects are removed from the model and thus consider that users and subjects are equivalent. In the process of developing HGABAC as EAP, some components are modified based on the requirements. *Operations (OP)* component of the conceptual HGABAC model has become *Policy*, which is a collection of policies defined for each operation in *OP*. All the assignment relations are ignored in the context of the HGABAC model, since it focuses only on the operational part of the model rather than the administrative part.

Users (U) is a set of individuals or automated entities (a system or a process) which make requests to access objects, where **Objects (O)** is a set of resources such as files, directories, applications, etc. Users and objects are associated with a specific set of **Attributes (Att)** respectively. These attributes reflect the properties and characteristics of users and objects. Each attribute is a function that takes an entity—a user, an object, a user group, or an object group, and returns one or more values from its range. The range of an attribute consists of a finite set of atomic values. All the attributes are set-valued in this model. **User attributes (UAtt)** is a set of user attributes for users and user groups. Similarly, **Object attributes (OAtt)** is a set of object attributes for object and object groups. **Operations (OP)** is a set of access rights such as *read*, *write*, etc. that can be performed on objects by users.

The set of user groups is **UG**, and the set of object groups is **OG**. These groups have a many to many hierarchical relationship (a partial order relation $\succeq_g$) among them, represented as *UGH* and *OGH* respectively. Thus, any senior group inherits all the attributes and values from the groups junior to it. For example, $g_1 \succeq g_2$ implies that g_1 is senior and inherits all the attributes and values

Table 3.3: *rHGABAC* Model with single-value EAP

I. Core Components
- *U, O* and *OP* are finite sets of users, objects, and operations respectively - *UG* and *OG* are finite set of user and object groups respectively - *UAtt* and *OAtt* are finite set of user attributes and object attributes functions respectively - For each *att* in *UAtt* $\cup$ *OAtt*, $Range(att)$ is a finite set of atomic values, where $Range(att_i) \cap Range(att_j) = \phi$ for $i \neq j$ - For each *uatt* in *UAtt*, *uatt* : $U \cup UG \rightarrow 2^{Range(uatt)}$, mapping each user and user group to a set of values in $Range(uatt)$ - For each *oatt* in *OAtt*, *oatt* : $O \cup OG \rightarrow 2^{Range(oatt)}$, mapping each object and object group to a set of values in $Range(oatt)$ - *directUg* : $U \rightarrow 2^{UG}$, mapping each user to a set of user groups - *directOg* : $O \rightarrow 2^{OG}$, mapping each object to a set of object groups - *UGH* $\subseteq$ *UG* $\times$ *UG*, a partial order relation $\succeq_{ug}$ on *UG* - *OGH* $\subseteq$ *OG* $\times$ *OG*, a partial order relation $\succeq_{og}$ on *OG*
II. Derived Components (Effective Attributes)
- For each *uatt* in *UAtt*, i) *effectiveUG*$_{uatt}$: *UG* $\rightarrow 2^{Range(uatt)}$, defined as *effectiveUG*$_{uatt}(ug_i)$ = *uatt*$(ug_i) \cup (\bigcup_{\forall g \in \{ug_j \mid ug_i \succeq_{ug} ug_j\}}$ *effectiveUG*$_{uatt}(g))$ ii) *effective*$_{uatt}$: $U \rightarrow 2^{Range(uatt)}$, defined as *effective*$_{uatt}(u)$ = *uatt(u)* $\cup (\bigcup_{\forall g \in directUg(u)}$ *effectiveUG*$_{uatt}(g))$ - For each *oatt* in *OAtt*, i) *effectiveOG*$_{oatt}$: *OG* $\rightarrow 2^{Range(oatt)}$, defined as *effectiveOG*$_{oatt}(og_i)$ = *oatt*$(og_i) \cup (\bigcup_{\forall g \in \{og_j \mid og_i \succeq_{og} og_j\}}$ *effectiveOG*$_{oatt}(g))$ ii) *effective*$_{oatt}$: $O \rightarrow 2^{Range(oatt)}$, defined as *effective*$_{oatt}(o)$ = *oatt(o)* $\cup (\bigcup_{\forall g \in directOg(o)}$ *effectiveOG*$_{oatt}(g))$
III. Policy Components
- *Policy*$_{op} \subseteq \{(ua_i, oa_j) \mid ua_i \in Range(uatt_k), oa_j \in Range(oatt_l)\}$, for $uatt_k \in$ *UAtt*, $oatt_l \in$ *OAtt*, and $op \in$ *OP* - $Policy = \{Policy_{op} \mid op \in OP\}$
IV. Authorization Function
- $is\_authorized(u : U, op : OP, o : O) = (\exists v_u \in$ *effective*$_{uatt_i}(u) \mid uatt_i \in$ *UAtt*$)$ and $(\exists v_o \in$ *effective*$_{oatt_j}(o) \mid oatt_j \in$ *OAtt*$)$, $[(v_u, v_o) \in$ *Policy*$_{op}]$

assigned to junior group g_2, in addition to its own directly assigned attributes and values. Users and objects can be assigned to zero or more user groups and object groups respectively. The function **directUg** takes a user and returns the set of user groups to which the user has been assigned, and **directOg** takes an object and returns the set of object groups to which the object belongs.

A user assigned to a user group gets all the attributes and values assigned to the group as well as inherited attributes and their values from junior groups. The effective attribute values of a user group *ug* are defined as *effectiveUG*$_{uatt}(ug)$ and comprise of directly assigned attribute values to the user group *uatt(ug)* and inherited attribute values from all the groups junior to it, i.e., effective attribute values of all the groups which are junior to *ug*. Hence, effective attribute values of a user *u*, defined as *effective*$_{uatt}(u)$, consists of attribute values directly assigned to the user *uatt(u)* and effective attribute values of all the groups assigned to the user. Similarly, effective values of object and object group attributes can be obtained [58]. These derived components and their formulas are shown in section II of Table 3.3.

In section III of Table 3.3, **Policy** is a set of policies where each policy is defined for a specific operation and is represented as *Policy*$_{read}$ for *read* operation. A policy for a particular operation is set of policy tuples defined for a single-value of user attribute and a single-value of object attribute. The function, $is\_authorized(u, op, o)$, authorizes a user *u* to perform an operation *op* on an object *o*, if there exists a policy tuple in *Policy*$_{read}$, which includes a user attribute value that belongs to the effective attribute values of user *u* and an object attribute value that belongs to the effective attribute values of object *o*.

• Extending *rHGABAC* with Attribute Hierarchies

Now, an attribute hierarchy is introduced to the *rHGABAC* model. Attribute hierarchy (AH), represented as ($\succeq_{att}$), is a partial order relation among attribute values in the range of an attribute *Range(att)*. It is defined in detail with an example as follows.

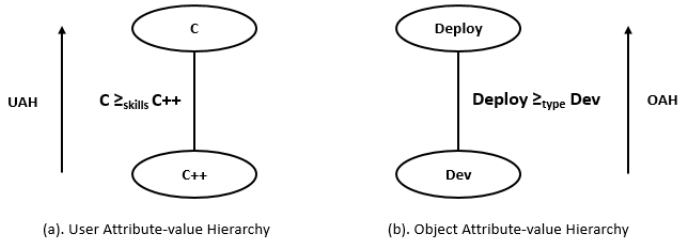


Figure 3.12: An Example of Attribute Hierarchy

Attribute Hierarchy

The concept of attribute hierarchy has been explored in literature in the context of attribute-based encryption by Li et al [76]. They presented hierarchical attribute-based encryption (HABE) mechanism where attributes can be classified in a tree structure based on their access control relationship in a system. An abstract idea of the encryption mechanism is that attributes being the nodes in a hierarchical tree, an ancestral node can derive its descendant's key, but converse is not allowed. However, the concept of attribute hierarchy in *rHGABAC* considers hierarchy among attribute values rather than attributes themselves. A similar concept has been presented by Biswas et al. [43] where an ABAC model (LaBAC$_H$), with one user attribute and one object attribute, have hierarchical relationship among attribute values.

The hierarchical relationship can be written as $\succeq_{att}$ and implies if a senior attribute value is assigned to a user or an object then all the junior attribute values are automatically acquired by that user or object through attribute-value inheritance. Figure 3.12 shows an example of attribute hierarchy. **User Attribute Hierarchy (UAH)** is depicted in Figure 3.12 (a) where *C* and *C++* belongs to the range of a user attribute named *skills*. There is a partial order relation between *C* and *C++*, represented as $C \succeq_{skills} C++$ which implies that value *C* is senior to *C++*. Therefore, if a user attribute value *C* is assigned to a user then it also acquires user attribute value *C++* and all its associated access privileges through attribute-value inheritance.

Similarly, in Figure 3.12 (b), an **Object Attribute Hierarchy (OAH)** is shown where *type* is an object attribute, and there is a hierarchical relation between two of its values *Deploy* and *Dev*,

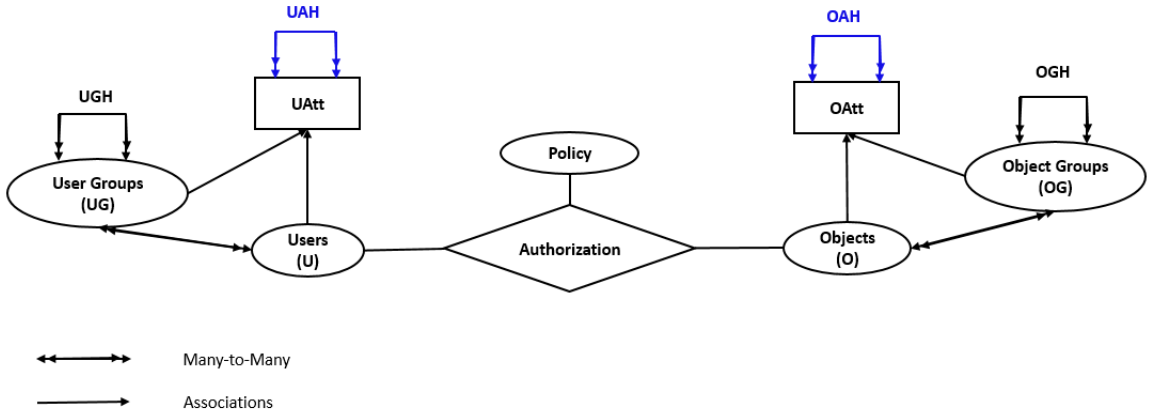


Figure 3.13: *rHGABAC* Model with Attribute Hierarchy

Table 3.4: *rHGABAC* with Attribute Hierarchy (AH) as single-value EAP

I. Additional Core Components
- $UAH_i \subseteq Range(uatt_i) \times Range(uatt_i)$, a partial order relation $\succeq_{uatt_i}$ on $Range(uatt_i)|uatt_i \in$ *UAtt* - $OAH_j \subseteq Range(oatt_j) \times Range(oatt_j)$, a partial order relation $\succeq_{oatt_j}$ on $Range(oatt_j)|oatt_j \in$ *OAtt*
II. Modified Authorization Function
- $is\_authorized(u : U, op : OP, o : O) = (\exists v_u \in$ *effective*$_{uatt_i}(u)|uatt_i \in$ *UAtt*$)$ and $(\exists v_o \in$ *effective*$_{oatt_j}(o)|oatt_j \in$ *OAtt*$)$, and $(\exists v_u \succeq_{uatt_i} v_u\prime)$ and $(\exists v_o \succeq_{oatt_j} v_o\prime)$,$[(v_u\prime, v_o\prime) \in$ *Policy*$_{op}]$

represented as *Deploy* $\succeq_{type}$ *Dev*. It implies that if an object is assigned object attribute value *Deploy* of object attribute *type*, then that object automatically gets its junior attribute value, *Dev*, assigned to it. The hierarchy also implies that the objects assigned to senior object attribute value, say *Deploy*, gets associated with access rights imposed on this attribute value as well as access rights imposed on all of its junior object attribute values, such as *Dev* here. Thus, attribute hierarchy facilitates policy management and attribute management. It also simplifies administrative task of assigning attributes and their values to users and objects and specifying access control policies in an ABAC model.

The *rHGABAC* model extended with partial order hierarchies among user attribute-values and object attribute-values is shown in Figure 3.13. The addition of attribute hierarchies requires modification of the authorization function. The additional core components and modified authorization

function are defined in Table 3.4. The authorization function is modified as it considers the effective attribute values of a user requesting access and a protected object through direct or group assignment along with attribute hierarchies, if any, among attribute-values of that user and object.

3.2.3 Enforcement Utilizing the Policy Machine and Authorization Engine

The implementation details of the authorization framework utilizing the PM and AE will be discussed in this section. PM allows to configure and implement different types of enumerated and logical-formula (with some restrictions) authorization policies. First, a generalized authorization architecture independent of any application or system is presented, and second, two variations of the *rHGABAC* model with the help of relevant use cases in the context of an enterprise organization are described and implemented in generalized authorization architecture.

The goal is to develop a general authorization framework which can be readily used by any application or system supporting RESTful service that is interested in implementing attribute-based access control policies. The architecture includes a PM Server as policy administration point (PAP) and policy decision point (PDP), and a PM Database used as policy information point (PIP) where all the access control data related to users, objects, their attributes, relations, etc. are stored.

A. Authorization Architecture

Figure 3.14 shows a generalized authorization architecture using the PM and AE. It consists of a PM Server which acts as a PAP and PDP. *Obligation* and *prohibition* relations for not considered in the architecture for now; therefore, there are no negative authorization policies. The PM Client is the authorization engine (AE) built in this research, which communicates with PM Server on one side and acts as a REST server for the applications on the other. Therefore, it also can be thought of as an interface between PM Server and applications using this architecture. The applications need not to be modified to make them PM aware, and instead can readily use PM for their security policy and authorization requirements. However, the applications need to support RESTful service to be

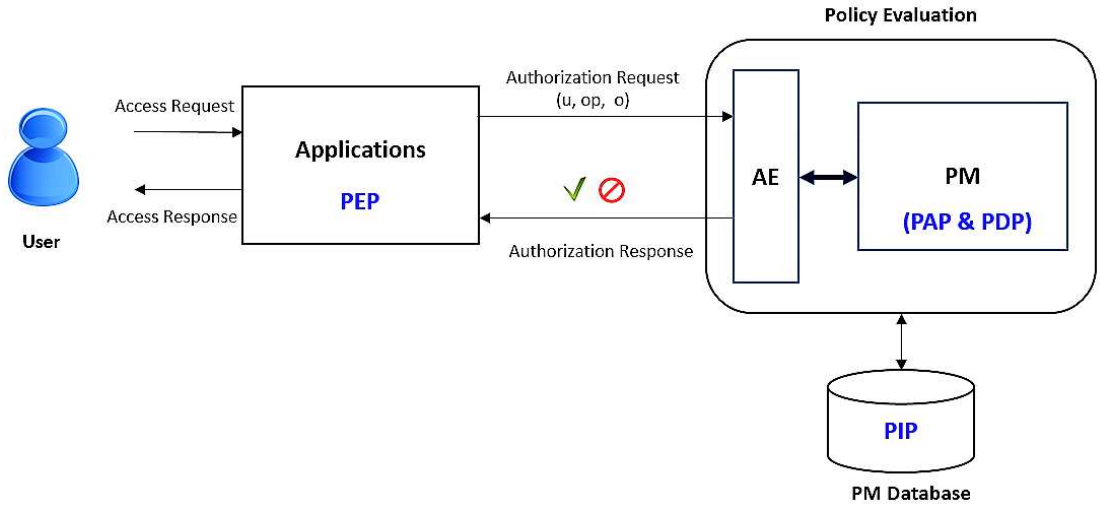


Figure 3.14: Authorization Architecture Utilizing PM and AE

```
stack@pm-app1:/$ curl -s http://192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
   "user":"user_1",
   "operation":"read",
   "object":"obj_Dev1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 3.15: Example Authorization Request and Response

able to make HTTP requests to AE. These applications are responsible for policy enforcement once the PM makes the authorization decisions. The PEP in the application would enforce the decisions on specific resources residing in the resource repository of these applications. This architecture is based on the assumption that both the PM and applications use the same identity management system, viz. Active Directory (AD) here, to store information about users, user groups, their attributes, etc.

Applications manage their objects and resources themselves including PEP and resources repository components as shown in the PM architecture. PM Server stores references to these objects in the PM database and policies are defined based on these objects and their attributes. When a user requests access to resources in an application, the application issues an HTTP request to AE for evaluating the policy which gets the authorization decision from the PM. The HTTP request would originate from a machine where the application is hosted. An authorization request comprises of

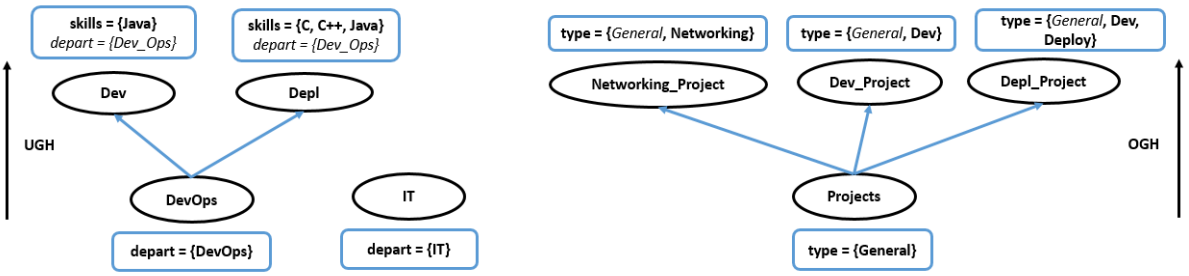
a *user* requesting access, an *operation* (e.g., read, write, execute), and the requested *object*. The request is implemented as an HTTP GET. An example of the authorization request and its response is shown in Figure 3.15. The request includes the authorization data in JSON format and gets a HTTP response with authorization decision returned in JSON format as well.

B. Policy Configuration and Setup

Policy Machine (PM) provides a robust and near complete access control framework with PDP, PAP, PIP, and PEP components included in one tool. Existing access control models such as DAC, MAC, and RBAC can be configured and implemented in PM [51, 52]. PM_{mini}, a bare minimimum version of PM with all the core elements and only two PM relations—*assignment*, and *association*, has been shown to be capable of representing attribute-based access control policies, such as *LaBAC*$_H$ [43]. In *LaBAC*$_H$ model, there is only one user attribute and one object attribute, and this scenario maps exactly with PM's inherent way of configuring access control policies.

The attributes in PM are containers for users and objects, whereas the attributes in ABAC policies are name-value pairs and the values of these attributes are used in the policy specification. The values of user and object attributes determine allowed accesses for a user on an object. Using a rich set of PM's capabilities and freedom to express access control policies in different ways, this dissertation identifies an intuitive method of configuring hierarchical attribute-based access control policies in the PM.

PM uses its user attributes and object attributes in access control policy specification. Different aspects of the *rHGABAC* model have to be mapped within this territory to configure and implement it in the PM. Therefore, user groups, user attributes, and user attribute-values of the model are represented as PM's user attributes, and object groups, object attributes, and object attribute-values as represented as PM's object attributes. PM's *assignment* relation is used to incorporate the hierarchical relationships in the model, taking advantage of the containment property of the attributes in PM. The policies for each operation are defined using *association* relation as defined for PM_{mini} [43]. The following subsections present use cases to provide a better understanding



(a). User Groups and their assigned user attributes and values

(b). Object Groups and their assigned object attribute and values

Figure 3.16: User and Object Groups with Associated Attributes

and demonstration of this mechanism.

C. Use Cases

This section discusses two variations of a use case, one with groups and group hierarchy, and other with the additional attribute-value hierarchy which is a modified version of the former. These use cases closely resemble a real-world enterprise scenario and are capable of representing different features of the *rHGABAC* model. These use case scenarios are configured and implemented in the PM using the generalized authorization architecture.

Enumerated and logical-formula policies are, in general, equally expressive. However, it is difficult to include negative attribute values in PM, such as *not a Manager* in a policy. For example, defining a policy, such as *a user with title IT_Manager and not with a title CTO is allowed to read objects with type Networking*, is a complicated task in the PM and requires the use of *prohibition* relation, constraints, and a combination of policies. However, a restricted HGABAC (*rHGABAC*) policy can be easily defined in PM using core elements and *assignment* and *association* relations.

- **Group Attribute and Group Hierarchy:**

This use case includes user groups, object groups, and hierarchy among groups, besides other ABAC components. A set of user groups, their attributes, and hierarchy among these groups is shown in Figure 3.16(a). Similarly, Figure 3.16(b) shows a set of objects groups, their attributes, and their hierarchical relationships. There are four user groups *DevOps, IT, Development* and *Deployment*, and there is a group hierarchy among *DevOps*, *Development*, and *Deployment*. *De-*

velopment and *Deployment* are senior to *DevOps* and inherit attributes and their values from it (*depart* and its value). They also have their own directly assigned attribute, i.e., *skills*, with specific values. *IT* group does not have any hierarchical relationship with other groups and has only one directly assigned attribute, i.e., $depart = \{IT\}$.

A similar object group hierarchy is shown in Figure 3.16(b) between four object groups, *Projects, Networking_Project, Dev_Project*, and *Depl_Project*. *Projects* is junior to all the other groups and has one object attribute *type* with one value *General* assigned to it. The senior groups inherit this object attribute and its value along with their directly assigned attribute values. For example, *Networking_Project* inherits object attribute *type* and its value *General* from *Projects*, and also has another value assigned to it as *Networking*. Thus, the effective attribute values of *Networking_Project* for attribute *type* is a set of all the values, directly assigned and inherited through the hierarchy, written as *{General, Networking}*.

In addition to two user attributes, namely *depart* and *skills*, there is one more user attribute, *title*, with range *{CTO, IT_Manager, DevOps_Manager}*. This user attribute is considered to be directly assigned to the users since titles are user-specific rather than being assigned to a group of users. There is only one object attribute, *type*, assigned to the object groups. Any object in an object group automatically gets all the attributes and its values assigned to it through that group. Similarly, users get user attributes and values assigned directly or through user groups based on their group membership.

For this use case, Figure 3.17 shows a graph similar to the PM element graph [52] presented in Chapter 2. A policy named Group Hierarchy Policy (GHP) is shown at level 0, user attributes and object attributes are shown at level 1. User attributes, their values, and user groups are on the left-hand side of the policy and object attributes, their values, and object groups on the right-hand side of the policy. User and object attribute values are placed at level 2, and these entities are assigned to each other using *assignment* relation. *Assignments* in the graph are represented as a directed edge, for example, *x ASSIGN y* is a directed edge from x to y.

User and Object groups are present at level 3 in this use case. If there is a hierarchy among

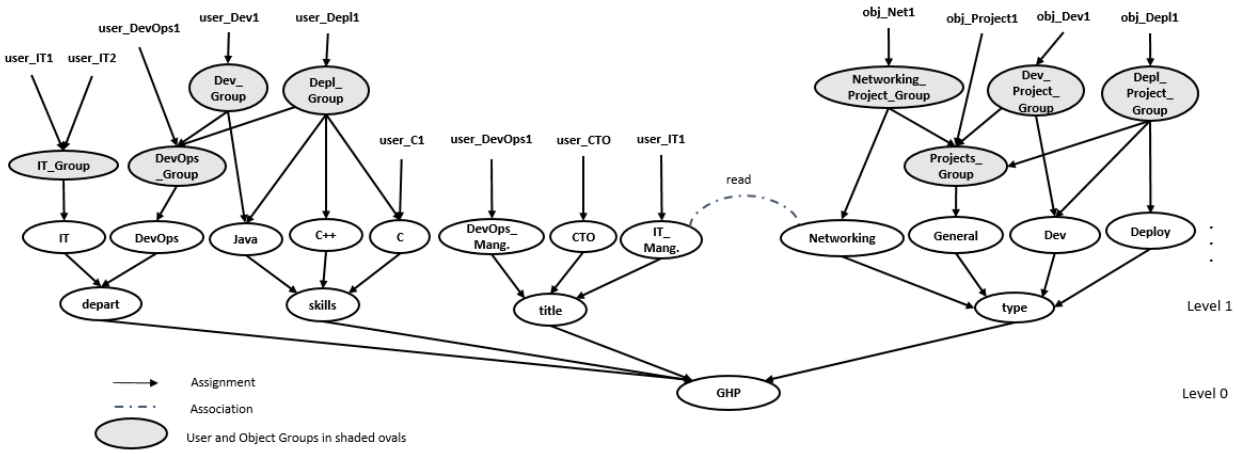


Figure 3.17: Group Hierarchy Policy Graph (Based on PM Graph Structure)

groups, then senior groups are shown one level above their junior groups, with constant increments in level for each level of hierarchy among groups. *DevOps_Group* is a junior group and is shown one level below the senior groups, *Dev_Group* and *Depl_Group*. The same applies to the object side where *Projects_Group* is shown one level below other senior object groups. The *containment* property of PM attributes enables to represent the hierarchical relationships between groups.

Users and objects are the leaf nodes of the graph and could appear at any level based on their assignment to user groups, user attribute values, and object groups, object attribute values respectively. A policy in the PM is defined using *associations* which comprise of a user attribute value, an operation and an object attribute value written as *(*ua_i, op, oa_j*)*. To keep the graph simple in Figure 3.17, we show only one association between *IT_Manager* (a user attribute value for *Title*) and *Networking* (an object attribute value for *type*) labeled as *read* using a dashed line. This association allows *user_IT1* to read *obj_Net1*. Therefore, the user assigned to this title, i.e., *IT_Manager* can read all the objects of type *Networking*.

This use case defines an access control policy for the *read* operation. Policies are specified based only on user attribute values and object attribute values. For each policy tuple in $Policy_{read}$, only a single value of user attribute and a single value of object attribute can be used in a single-value EAP. A policy for read operation, based on Figure 3.17, is specified as follows.

i. A user who is an *IT_Manager* or works in the IT department can read objects of type *Net-*

Table 3.5: Policy for *Read* Operation with Group Hierarchy

$Policy_{read}$	
User Attribute Values	Object Attribute Values
IT_Manager	Networking
IT	Networking
DevOps_Manager	Dev
Java	Dev
DevOps_Manager	Deploy
Java	Deploy
C	Deploy
C++	Deploy
CTO	General

```
stack@pm-app1:/$ curl -s http:// 192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
  "user":"user_IT2",
  "operation":"read",
  "object":"obj_Net1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 3.18: Sample Authorization Request and Response

working.

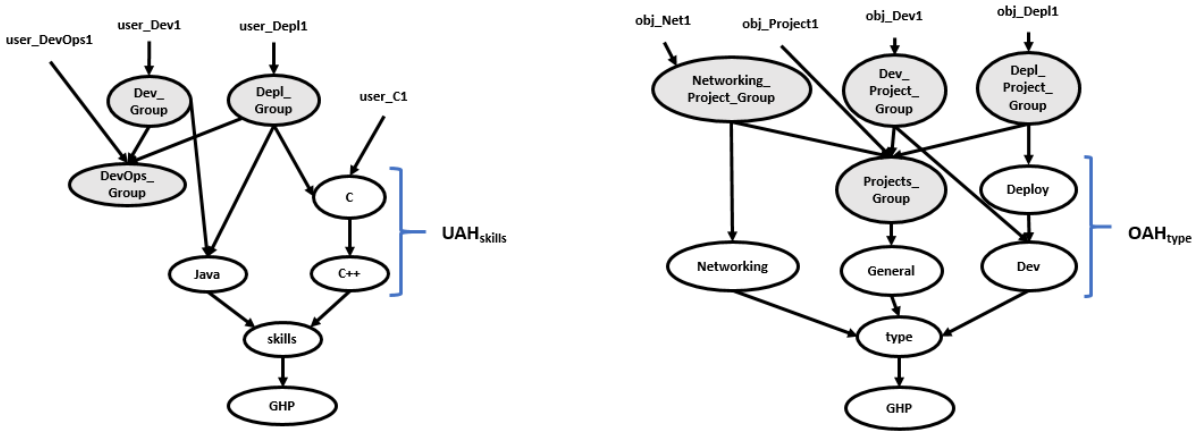
ii. A user who is a *DevOps_Manager* or has *Java* skill can read objects of type *Dev*.

iii. A user who is a *DevOps_Manager* or has *Java* or *C* or *C++* skill can read objects of type *Deploy*.

iv. A user who is a *CTO* can read objects of type *General*.

The fourth policy statement incorporates a powerful policy, where a CTO can read all the objects in the above scenario due to the attribute inheritance achieved through object group hierarchy. All the groups have type *General* assigned to them through object group hierarchy and attribute inheritance. Thus any object assigned to any one of these groups can be read by a CTO. Therefore, many implied policies can be derived via one policy tuple.

Table 3.5 presents a set of policy tuples for $Policy_{read}$. Each row of the table represents an association defined in the PM. An authorization request involving user *user_IT2*, operation *read*



(a). Subgraph Showing Attribute Hierarchy in *skills* Attribute

(b). Subgraph Showing Attribute Hierarchy in *type* Attribute

Figure 3.19: Attribute Hierarchy

and object *obj_Net1*, along with its response is shown in Figure 3.18. This request is granted based on the second row of Table 3.5.

The importance and benefits of group attributes and hierarchy can be realized when there are numerous users and objects in the system, and attributes need to be assigned to each one of them. This use case presents a limited number of users and objects for the sake of clarity. However, in a real-world scenario, there are hundreds, thousands, or even millions of users and objects. In such a scenario, *rHGABAC* could be an effective solution for access control requirements with simplified attribute assignment and administration, and better policy management.

- **Attribute Hierarchy:**

In enumerated authorization policies, an inherent problem is the exponential size of a sophisticated access control policy and its associated space requirements. The number of policy tuples in an authorization policy, for an operation, would increase exponentially with the large size of ranges of user attributes and object attributes. A policy consists of a set of policies defined for different operations (e.g., *read, write,* etc.) in a system which further worsens the problem. An interesting solution to this problem could be to realize hierarchical relationships between the values of user attributes and object attributes.

With attribute hierarchy, a single policy tuple defined in the policy can imply many policy tuples

Table 3.6: Policy for *Read* Operation with Group and Attribute Hierarchy

$Policy_{read}$	
User Attribute Values	Object Attribute Values
IT_Manager	Networking
IT	Networking
DevOps_Manager	Dev
Java	Dev
C++	Deploy
CTO	General

through partial order relations among user and object attribute values. This subsection introduces attribute hierarchies among ranges of one user attribute (*skills*) and one object attribute (*type*) from the above use case. The subgraph for both *skills* and *type* are shown in Figure 3.19(a) and 3.19(b) respectively. In the use case graph of Figure 3.17, *skills* has all three values assigned to it at the same level. However, in subgraph of Figure 3.19(a) *C* is assigned to *C++* and $C \succeq_{skills} C++$. This means that attribute value *C* is senior to *C++* and gets all the properties and capabilities of *C++*. Therefore, any accesses allowed for *C++* would be permitted for *C* as well.

In Figure 3.19(b), the *type* attribute is shown with a partial order hierarchy between its values *Deploy* and *Dev*, written as $Deploy \succeq_{type} Dev$ where *Deploy* is senior to *Dev*. Any *association* involving junior attribute value will also be implied on a senior attribute value. In other words, *Deploy* will inherit all the associations applied to *Dev*.

Based on attribute hierarchies introduced in *skills* and *type*, the *associations* defined in PM are changed. The updated policy tuples for $Policy_{read}$ are shown in Table 3.6. Three policy tuples were removed since they were implied through attribute hierarchy. Two tuples *(DevOps_Manager, Deploy)* and *(Java, Deploy)* were removed since they were implied through hierarchy between *Deploy* and *Dev*. Similarly, *(C, Deploy)* is removed as its implied through hierarchy between *C* and *C++*. This clearly shows how hierarchical attributes can be exploited in reducing the size of enumerated policies. It also contributes in simplifying administrative tasks by reducing the number of direct policy tuples to be defined in a policy.

An authorization request for user *user_C1* (assigned to *C*), operation *read* and object *obj_Depl1*

```
stack@pm-app1:/$ curl -s http:// 192.168.1.0:9000/echoGet -X GET -d
'{"type":"hierarchical",
 "user":"user_C1",
 "operation":"read",
 "object":"obj_Depl1"
}'
{"access":"granted"}
stack@pm-app1:/$
```

Figure 3.20: Sample Authorization Request and Response

(assigned to *Deploy*), along with its response is shown in Figure 3.20. Even though there is no direct policy tuple defined between *C* and *Deploy* attribute values, this request is granted through an implied policy tuple based on effective user attribute values of the user and effective object attribute values of the object.

The partial order relationship among groups and attributes depends on how they are realized while defining authorization policies. Some hierarchical ordering is intuitive, whereas others may not be so evident. It is for security architects and administrators to design these hierarchies based on appropriate techniques, such as attribute mining or attribute engineering, to define access control policies in the most efficient way. However, these techniques are outside the scope of this dissertation.

D. Evaluation

This section conducts experiments for measuring the policy evaluation times for different types of attribute-based policies (based on above use cases) in AE and PM. AE receives an authorization request when a user requests access on an object in an application. It communicates to the PM to get authorization decisions. For the evaluation, the time taken by AE to communicate to the PM and evaluate authorization decisions was measured for three type of access control policies: *Role-Centric ABAC*, *rHGABAC without AH*, and *rHGABAC with AH*. Table 3.7 presents the average policy evaluation time for each of these policies. The average policy evaluation time using AE and the PM is shown in the second column, while the types of access control policies are shown in the first column.

For the experiments, initially, each type of policy is setup in the PM. Then multiple autho-

Table 3.7: Average Policy Evaluation Time for ABAC Policies

Policy	Avg. Time (ms)
Role-Centric ABAC	26.04
rHGABAC	27.04
rHGABAC with AH	26.57

rization requests from a user are executed for each one of them to obtain averages of their policy evaluation times. The average policy evaluation time for different policies is very close to each other with a maximum difference of 1 millisecond. The results were consistent with the initial hypothesis of this research since the average policy evaluation times is expected to be similar, mainly due to the way ABAC policies are configured in the PM and implemented in the authorization architecture.

The average policy evaluation time for *rHGABAC with AH* is slightly less than *rHGABAC*. The *rHGABAC with AH* model is capable of representing complex and large policies in a concise way using the hierarchies among attribute values. It is also evident in the use case discussed in the Use Case section above. A concise policy in the PM implies a compact PM policy graph with fewer associations between entities or elements. Both *rHGABAC* and *rHGABAC with AH* policy defined in the PM has the same set of users, user groups, user attributes, objects, object groups, object attributes; however, the number of association relations (policy tuples) varies which results in a disperse policy tree for *rHGABAC* compared to a concise or small policy tree for *rHGABAC with AH*. Thus, this could be one of the possible reasons for slightly faster policy evaluation time in *rHGABAC with AH* policy, since the small number of association relations defined in the PM would involve less internal evaluations to evaluate an access control decision.

A role-centric ABAC is a special type of attribute-based access control policy (presented in the first section of this chapter) which includes roles and attributes in the policy specifications and policy decisions. Although this policy had the lowest average of policy evaluation time in the experiments, the time would increase exponentially if there are numerous roles that a user can be assigned. It is a combination of role-based and attribute-based access control policy and would require a more detailed investigation against similar combinational access control policies.

However, this work does not focus on such in-depth analysis of combinational models, hence is outside the context of this dissertation.

Overall, the initial investigation shows that the average policy evaluation times are comparable across different types of ABAC models. The use cases and their implementation demonstrate that the *rHGABAC* model can be conveniently implemented utilizing the PM and AE, and can be applied in real-world applications and systems. However, the enforcement framework in the applications (e.g., OpenStack) utilizing the PM need to be designed accordingly in order to optimize policy evaluation and policy enforcement times.

3.3 Related Work

This section briefly reviews the prior efforts which have been made on adding attributes to the OpenStack access control and authorization framework. The applicability of ABAC in OpenStack has been studied in different scenarios. Approaches to include attributes in OpenStack for cloud federation and federated identity management are discussed by Chadwick et al. [46] and by Lee and Desai [75]. Pustchi and Sandhu [88] discuss the application of ABAC to enable collaboration between tenants in a cloud IaaS platform. In contrast, the UAE-OSAC model is focused on authorization in OpenStack within a single tenant.

Simialrly, a unified ABAC model is presented in [67] by Jin et al. that can be configured to do mandatory, discretionary and role-based access control, and demonstrated an OpenStack implementation [68] as a replacement for the native OpenStack RBAC. However, the UAE-OSAC model is specifically designed to incorporate OpenStack's existing RBAC model with an ABAC extension. A formal role-centric attribute-based access control (RABAC) model has been proposed by Jin et al. [69], along with XACML [32] profiles for this model. The role-centric ABAC extension for OpenStack developed here, in contrast, includes only user attributes as a first step leaving object attributes for future work. In cloud environments the objects are created on-demand, hence cannot be specified in the policy during policy configuration [105].

Besides, the authorization architecture is different and more flexible based on the PM and AE.

PM is an open-source and freely available software by NIST [25] that facilitated in the development of a customized authorization engine component in this research. PM's flexibility of specifying different types of access control policies beyond attribute-based and its capabilities made it a desirable choice for it to be used in the enforcement framework.

CHAPTER 4: ABAC FOR AWS INTERNET OF THINGS

This chapter investigates a major commercial cloud-IoT platform, AWS IoT [3], provided by Amazon Web Services (AWS) [1]. It develops a formal access control model for AWS IoT called AWS-IoTAC. AWS-IoTAC builds upon the AWS Access Control (AWSAC) model developed by Zhang et al. [115] for AWS access control in general. The AWS-IoTAC model is demonstrated using a smart-home use case implementation in the AWS IoT platform. Inspired by the use case, this dissertation identifies the need of a fine-grained access control mechanism for IoT and proposes some Attribute-Based Access Control (ABAC) extensions to the AWS-IoTAC model for enhancing the flexibility of access control in AWS IoT. Major sections of this chapter are based on the following publication [40] with some revisions and modifications.

- Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. Access Control Model for AWS Internet of Things. In *International Conference on Network and System Security*, pages 721-736. Springer, 2017.

4.1 AWS IoT Access Control (AWS-IoTAC) Model

Internet of Things (IoT) refers to a network of internet-enabled physical devices or things, and their underlying communication with each other and other internet-enabled devices and systems [18]. With the advent of technology and ubiquitous internet, devices besides computers, laptops, and smartphones, are becoming smarter along with internet connectivity such as televisions, vehicles, thermostats, bulbs, watches, etc. to make human life more comfortable and convenient.

IoT is a pervasive concept today. The building blocks of IoT, i.e., the IoT devices are resource-constrained with limited computational power, storage, and memory space. Therefore, to support rapidly broadening IoT infrastructure, Cloud Computing has become an enabling technology. The integration of cloud and IoT has been suggested as a viable IoT architecture in the literature [33, 37, 44, 79, 85, 86, 90, 91]. In general, a Cloud-Enabled Internet of Things (CE-IoT) architecture comprises a centralized Cloud that provides computation, storage, and analytic power to

resource-constrained IoT devices. In CE-IoT, the burden of secure connection and communication is subsidized by the Cloud.

Security is an essential requirement for IoT, especially as deployments grow. The number of connected devices is increasing exponentially. As expected by Gartner, there will be more than 20 billion connected devices by 2020 [8]. This rapid growth in IoT has given rise to an attractive and new attack surface. Access control is an essential component of security solutions for IoT. There has been significant research done on access control models for IoT in academia, while industrial deployment of several cloud-enabled IoT (CE-IoT) platforms have already been introduced.

One such instance of CE-IoT platform is AWS IoT, an IoT service introduced by Amazon Web Services (AWS), one of the largest Cloud services provider. This dissertation studies and explores the AWS IoT platform, and develops an abstract access control model for it, which is called the AWS-IoTAC model. This model is developed by extending AWS cloud's formal access control (AWSAC) model, previously published in the academic literature [115], to incorporate the IoT specific components. The AWS-IoTAC model is abstracted from AWS IoT documentation and has been formalized based on AWSAC definitions.

4.1.1 AWS-IoTAC: Motivation

IoT has received considerable attention in both industry and academia in recent years. Accordingly, many access control models for IoT have been proposed. Ouaddah et al. [84] provide a recent survey of these. Meanwhile, dominant cloud providers, such as Amazon Web Services (AWS) [1], Microsoft Azure [21], and Google Cloud Platform (GCP) [11], have built upon their existing cloud services and resources to launch IoT services. Azure and GCP utilize some customized form of role-based access control (RBAC) [54,93] with predefined roles and groups for their access control requirements in the cloud. GCP uses RBAC for its IoT solutions authorization [12]. AWS uses a policy-based access control mechanism for its Cloud and IoT services [1, 3]. Unlike Azure cloud, Azure IoT has adopted policy-based access control to specify IoT authorizations [7]. However, as yet there is no consensus on a formal access control model for CE-IoT.

The IoT services require new concepts beyond basic access control in the cloud. Therefore, the primary motivation behind this research is to investigate a real-world CE-IoT platform and develop a formal access control model for it which is still lacking. In this research, the real-world CE-IoT platform considered is AWS IoT, leading to an abstract access control model for it known as AWS-IoTAC. This model is abstracted from dispersed AWS IoT documentation available, along with detailed exercises on this service to validate the understanding of the IoT functionality. This model serves as an initial attempt towards the development of standard access control models for CE-IoT.

While developing an access control model for IoT, it is useful to conceptualize the model in the context of a well-defined IoT architecture. A layered access control oriented (ACO) architecture for cloud-enabled IoT has been proposed by Alshehri and Sandhu [35]. Mapping of different entities of the AWS-IoTAC model with the four layers of ACO architecture is presented to underscore the relevance of the model with a CE-IoT architecture specially designed from an access control perspective. A smart-home use case is demonstrated and configured in the AWS IoT platform to depict the applicability of the AWS-IoTAC model in addressing IoT authorizations in AWS.

Currently, most of the CE-IoT platforms utilize some customized form of RBAC, but RBAC by itself is insufficient to address the dynamic requirements of IoT. With billions of connected devices in the near future, it will become inevitable for IoT to adopt a flexible access control model, such as attribute-based access control (ABAC) [63, 67], for meeting dynamic access control requirements of the IoT services. In ABAC, attributes (properties), represented as name-value pairs, of users and resources are utilized to determine user accesses on resources or services. AWS IoT supports a partial form of ABAC with attributes for the IoT devices, however, the use of these attributes in access control policies is limited. Therefore, we propose ABAC enhancements to our (AWS-IoTAC) model for incorporating a complete form of ABAC in it.

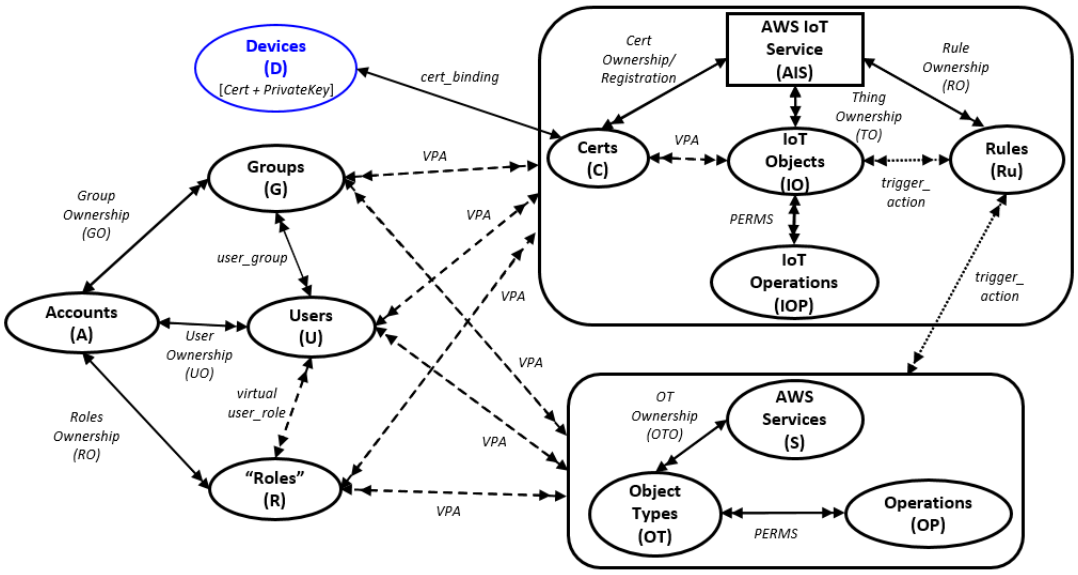


Figure 4.1: AWS IoT Access Control (AWS-IoTAC) Model within a Single Account

4.1.2 AWS-IoTAC: Model and Definitions

An access control model for AWS IoT, a cloud-enabled IoT platform, involves different entities in the IoT space and should define how these entities are authorized to interact with each other securely. This dissertation incorporates the entities involved in access control and authorization in the AWS IoT service into the AWSAC model so as to develop the AWS-IoTAC model. AWS-IoTAC is based on meticulous exploration of the extensive documentation on AWS IoT and hands-on experiments on this service.

The AWS-IoTAC model is shown in Figure 4.1 along with its different components. Since it is developed on top of the AWSAC model, it consists of all the components and relations of AWSAC with additional set of components and relations associated with the AWS IoT service. A review of AWSAC model has been presented in Chapter 2. Table 4.1 gives the formal definitions for the AWSAC model. **A, U, G, R, S, OT,** and **OP** are finite sets of accounts, users, groups, roles, services, object types, and operations in AWS respectively. The ownership relations **UO, GO,** and **RO** represents a mapping of users, groups, and roles to a specific account respectively. The set of permissions **PERMS** is defined based on object types with pertinent operations on these object types. The relation $user\_group$ represents a many-to-many relationship between users and groups

Table 4.1: AWSAC Model Components [115]

Definition 1.
- A, U, G, R, S, OT and OP are finite sets of accounts, users, groups, roles, services, object types, and operations respectively
- User Ownership $(UO) : U \rightarrow A$, is a function mapping a user to its owning account, equivalently a many-to-one relation $UO \subseteq U \times A$
- Group Ownership $(GO) : G \rightarrow A$, is a function mapping a group to its owning account, equivalently a many-to-one relation $GO \subseteq G \times A$
- Role Ownership $(RO) : R \rightarrow A$, is a function mapping a role to its owning account, equivalently a many-to-one relation $RO \subseteq R \times A$
- Object Type Ownership $(OTO) : OT \rightarrow S$, is a function mapping an object type to its owning service, equivalently a many-to-one relation $OTO \subseteq OT \times S$
- $PERMS = OT \times OP$, is the set of permissions
- Virtual Permission Assignment (VPA): $VPA \subseteq (U \cup G \cup R) \times PERMS$, is a many-to-many virtual relation resulting from policies attached to users, groups, roles and resources
- user_group $\subseteq U \times G$ is a many-to-many mapping between users and groups where users and groups are owned by the same account
- virtual user_role (VUR): $VUR \subseteq U \times R$ is a virtual relation resulting from policies attached to various entities (users, roles, groups), where users use AssumeRole action to acquire/activate a role authorized in VUR

within a single account.

An AWS policy, a JSON document, defines permissions on resources (services, object types). It can be attached to users, groups, and roles, as well as resources itself. Based on the policy assignment to various entities, a many-to-many virtual relationship is defined as **Virtual Permission Assignment (VPA)**. Similarly, a virtual relationship $virtualuser\_role$ results when a user uses the *AssumeRole* action to activate an AWS "role" for itself and acquire permissions based on policies attached to the user, roles, and groups. In addition to the AWSAC components and relations, the AWS IoT service has its specific components and relations which are incorporated and formally defined in the AWS-IoTAC model. The additional or modified components and relations are formally defined in Table 4.2 and discussed below.

There are six additional components in the AWS-IoTAC model. **AWS IoT Service (AIS)** is the new IoT service in AWS. It owns different entities to support IoT devices and their underlying au-

Table 4.2: AWS-IoTAC Model – Additional Components and Relations

Definition 2.
- AWS IoT Service (AIS) is one of the Services(S) in AWS
- C, D, IO, IOP, and Ru are finite sets of X.509 certificates, physical IoT devices, IoT objects, IoT operations, and rules defined in the rules engine of AIS respectively
- Cert Ownership/Registration $(CO) : C \rightarrow AIS$, is a function mapping a certificate to its owning service (AIS), equivalently a many-to-one relation $CO \subseteq C \times AIS$
- Rules Ownership $(RO) : Ru \rightarrow AIS$, is a function mapping a rule to its owning service (AIS), equivalently a many-to-one relation $RO \subseteq Ru \times AIS$
- Thing Ownership $(TO) : IO \rightarrow AIS$, is a function mapping the IoT objects to its owning service (AIS), equivalently a many-to-one relation $TO \subseteq IO \times AIS$
- $PERMS = OT \times OP$, is the set of permissions (including IoT permissions)
- Virtual Permission Assignment (VPA): $VPA \subseteq (U \cup G \cup R \cup C) \times PERMS$, is a many-to-many virtual relation resulting from policies attached to users, groups, roles, certificates, and resources
- cert_binding $\subseteq C \times D$ is a mutable one-to-one relation between X.509 certificate and IoT devices within a single account
- trigger_action $\subseteq Ru \times (IO \times S)$ represents a many-to-many mapping between rules and IoT objects and AWS services on which a rule triggers action(s)

thorization in the cloud. The AWS-IoTAC model represents AIS as a separate entity to emphasize its importance and clearly show other components and relations associated with it. The rectangular box of AIS emphasizes its singleton existence in AWS. **Certs (C)** is a set of X.509 certificates [31], issued by a trusted entity, the certificate authority (CA). AIS can generate X.509 certificates for the IoT clients, or allow the use of certificates created by the clients as long as they are signed by a registered CA in the AWS IoT service. Certs are used by MQTT [22] based clients (IoT devices, applications) to authenticate to AIS. MQTT, an OASIS standard, is a machine-to-machine (M2M) lightweight publish/subscribe messaging protocol, specially designed for constrained devices [22]. Currently, AWS IoT supports MQTT and HTTP protocol for IoT communications.

Devices (D) represent a set of connected IoT devices, such as sensors, light bulbs. They can exist independent of AIS, thus are shown in a different color in the model. A valid X.509 certificate

and its private key need to be copied onto the device, along with a root AWS CA certificate before authentication and establishment of a secure communication channel with the AWS IoT service. The certificates to devices associations are done through the $cert\_binding$ relation. In the AWS IoT platform, one certificate can be attached to many things/devices. Similarly, many certificates can be copied onto one IoT device. However, AWS-IoTAC assumes that $cert\_binding$ is a one-to-one association between devices and certificates for better authorization management, and is mutable so can be changed by an administrator in cases of certificate expiry or revocation. In AWS IoT, the access control policies are attached to certificates and are enforced on physical IoT devices associated with these certificates.

IoT Objects (IO) represent virtual IoT objects in the cloud. *Virtual objects* are the digital counterparts of real physical devices, or standalone logical entities (applications) in the virtual space [82]. In AWS IoT, a *Thing* and a *Thing Shadow* represent the IoT objects which are the virtual representation of real physical IoT devices in the cloud. For each IoT device, the model assumes that there is at least one *thing* with its *thing shadow* instantiated in the cloud, which provides a set of predefined MQTT topics/channels (associated with this device) to allow interaction with other IoT devices and applications, even when the device is offline/disconnected. *Thing shadow* maintains the identity and last known state of the associated IoT device, as well as stores a future desired state of the device.

IoT Operations (IOP) are a set of operational operations defined for IoT service and do not include the administrative operations, such as create things, certificates, etc. The basic set of IoT operations can be categorized based on the communication protocols used by IoT devices and applications to communicate with the AWS IoT service. For MQTT clients, four basic IoT operations are available: i) *iot:Publish* allows devices to publish a message to an MQTT topic, ii) *iot:Subscribe* allows a device to subscribe to the desired MQTT topic, iii) *iot:Connect* allows an MQTT client to connect to the AWS IoT service, and iv) *iot:Receive* allows devices to receive messages from subscribed topics. Similarly, for HTTP clients, *iot:GetThingShadow* allows to get the current state of a thing shadow, *iot:UpdateThingShadow* allows to send messages to update/change

the state of a thing shadow, and *iot:DeleteThingShadow* deletes a thing shadow. Whenever a device or an application sends a message to a virtual thing in the cloud, a new thing shadow is automatically created, if one does not already exist.

Rules (Ru) are simple SQL statements which trigger predefined actions based on the condition defined in the rule. A rule receives data from a device/thing and triggers one or more actions. The actions route the data from one IoT device to other IoT devices, or to other AWS services. Each rule must be associated with an IAM (Identity and Access Management) role which grants it permissions to access required IoT objects and AWS services on which actions are triggered. The relation $trigger\_action$ represents a many-to-many mapping between rules and IoT objects and AWS services on which the rule triggers action(s).

The access control policies in AWS have been modified to include IoT operations and resources, and are thereby named as IoT policies. AWS IoT utilizes both IoT policies and IAM policies to assign specific permissions to IoT devices, IAM users, and IoT applications. Consequently, **Virtual Permission Assignment (VPA)** has been updated to include the IoT policies, and these policies are attached to X.509 certificates. The policy attached to a certificate is enforced on the device which uses that certificate to connect and authenticate to the AWS IoT service. One policy can be attached to multiple certificates, or multiple policies can be attached to one certificate.

All the components and relations of the AWS-IoTAC model are defined within the scope of a single AWS account. Cross-account authorizations are outside the scope of this research. The components and relations of AWS-IoTAC are based on current capabilities of the AWS IoT service. Although there are many other components and relations associated with the AWS IoT service, this model encompasses the most important ones from an access control perspective.

4.1.3 AWS-IoTAC Mapping in ACO Architecture

This section shows the relevance of the AWS-IoTAC model to the Access Control Oriented (ACO) architecture for IoT presented by Alshehri and Sandhu [35]. Figure 4.2 depicts a mapping of different entities of the AWS-IoTAC model with the ACO architecture. Different entities map to

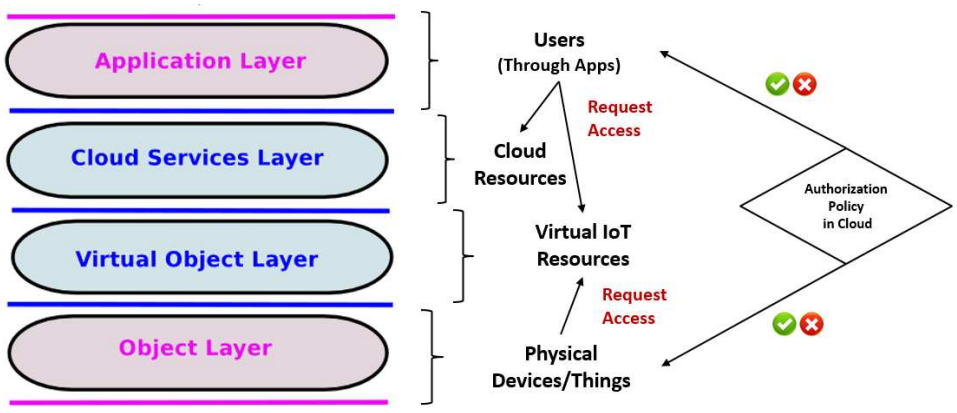


Figure 4.2: AWS-IoTAC Entities Mapping to ACO Architecture for CE-IoT

different layers of the ACO architecture. Physical devices or things exist at *Object layer*, and virtual IoT things or resources map to the *Virtual Object layer*. All the AWS cloud services and resources are at *Cloud Service layer*, and users and applications interacting with the Cloud and IoT devices exist at the *Application layer*. The authorization policies are defined in the cloud. These policies enforce access control decisions for physical devices and applications (used by users) trying to access cloud and virtual IoT resources. AWS-IoTAC is generally compatible with the ACO architecture.

4.2 A Smart Home Use Case in AWS IoT

This section presents a smart-home use case where a thermostat and two light bulbs are controlled through the AWS IoT service based on sensor inputs. The main focus here is on interactions between IoT devices through the cloud. (A more complex example would involve different users and applications also interacting with IoT devices.) It demonstrates how the access control and authorization between various components are configured based on the AWS-IoTAC model.

4.2.1 Use Case Setup and Configuration

Figure 4.3 shows different connected devices, virtual things/objects, and AWS Cloud and IoT Services involved in the use case. First, an AWS account is created to setup the use case in the AWS IoT service. Using AWS IoT management console, one virtual object (*thing*) for each physical

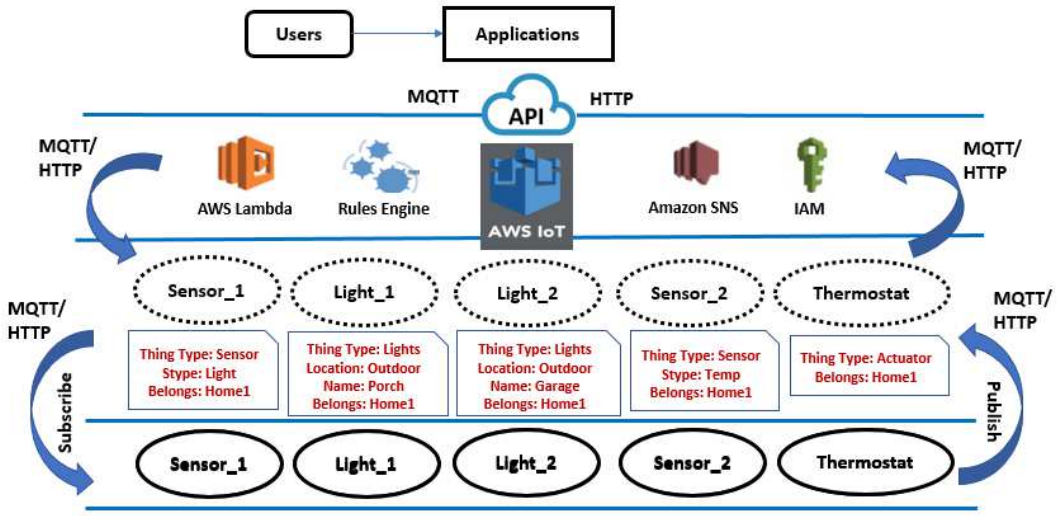


Figure 4.3: Smart-Home Use Case Utilizing AWS IoT and Cloud Services

device—two sensors, one thermostat, and two light bulbs were created. A *thing* can have a *thing type* that stores configuration for similar things, and *thing attributes* (key-value pairs) representing properties of individual IoT devices.

For example, *Sensor_1* has a *Sensor* thing type and has two attributes *SType* (sensor type) and *Belongs* (belongs to). The values for these attributes are set during thing creation. The X.509 certificates for each IoT thing/device must be created using "one-click certificate creation" in the AWS IoT console. Then, appropriate authorization policies are defined and attached to the certificates. After desired policy attachment to the certificate, it is attached to a virtual thing and is copied onto its corresponding physical device along with its associated private and an AWS root CA certificate. The CA certificate specifies the identity of the server, viz., AWS IoT server in this case. A device certificate is used during device authentication and specifies its authorization based on attached policies. The lights and thermostat devices are simulated using AWS SDKs (Node.js) [6] provided by AWS, and sensors are simulated as MQTT clients using MQTT.fx tool [22]. All these devices use MQTT protocol to communicate to the AWS IoT service with Transport Layer Security (TLS) [28].

Based on the use case scenarios, the *rules engine*, a part of the AWS IoT platform, is utilized to define rules and trigger desired actions. The actions include a *Lambda function* and notification to

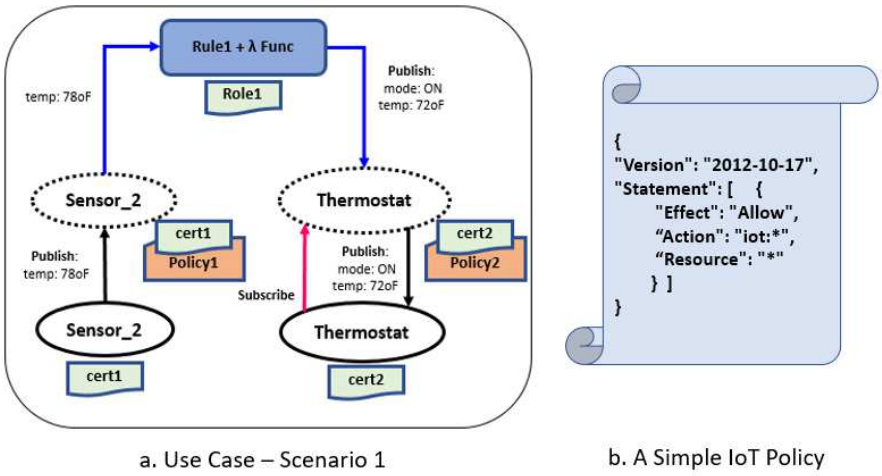


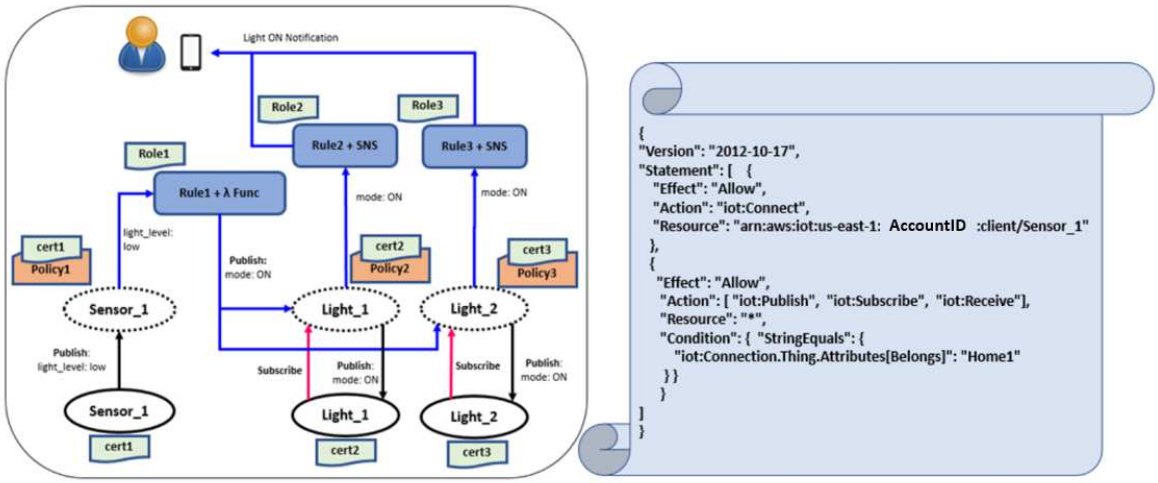
Figure 4.4: Smart-Home Use Case Scenario 1

users by sending text messages through *Amazon Simple Notification Service (SNS)*. For each rule, an IAM "role" is associated with it to authorize the rule to access required AWS and AWS IoT services and resources.

4.2.2 Use Case Scenarios

This section discusses two scenarios of the use case and their relevant authorization aspects.

A. **Scenario 1:** This scenario involves a temperature sensor and a thermostat and is depicted in Figure 4.4(a). A temperature sensor *Sensor_2* (shown in solid oval) senses the temperature and sends data to its thing shadow, *Sensor_2* (shown in dotted oval), in the AWS IoT platform. Based on *Sensor_2* data, a rule (Rule1) triggers a lambda function to change the state of the *Thermostat* by publishing an update message to its thing shadow (shown as the dotted oval). If the environment temperature is greater than 78 degrees Fahrenheit, then the rule invokes a lambda function that publishes a message on *Thermostat* thing shadow to turn on the thermostat and set its temperature to 72 degrees Fahrenheit. The physical thermostat (shown in solid oval) has subscribed to its shadow topics, hence, receives the update message and syncs its state with its thing shadow. For this scenario, a simple authorization policy is defined for both *Sensor_2* and *Thermostat*, as shown in Figure 4.4(b). It allows an entity



a. Use Case – Scenario 2 b. A Fine-grained IoT Policy

Figure 4.5: Smart-Home Use Case Scenario 2

to do any IoT operation (e.g., publish, subscribe) on any resource within a specific account and selected region in the AWS cloud. The policy is attached to the X.509 certificates which are copied onto the corresponding physical IoT devices (*Sensor_2* and *Thermostat*). In this example, the physical devices have full IoT access on all the resources in AWS IoT.

B. **Scenario 2:** A more comprehensive scenario with a fine-grained authorization policy is presented in Figure 4.5. A light sensor, *Sensor_1*, monitors the light level of the environment and turns on outdoor lights, *Light_1* and *Light_2*, when the light level is low. When the lights are turned on, users (owner or resident) of the home get a text notification about the state change of the lights. For this scenario, a more restrictive policy is defined for *Sensor_1* which utilizes thing attributes in the *Condition* section of the policy. The policy is shown in Figure 4.5(b), and comprises two policy statements—first to authorize a client to connect to AWS IoT only if its client ID is *Sensor_1*, and second to allow IoT publish, subscribe, and receive operations on all resources only if the client requesting access has a thing attribute *Belongs* with a value *Home1*. This policy employs thing attributes in making access control decisions. Thing attributes, as shown in Figure 6.6, represent the characteristics of IoT things/devices.

Currently, AWS IoT policy supports thing attributes of only those clients (devices/things) which are requesting access to resources in the AWS IoT service. A useful scenario would be

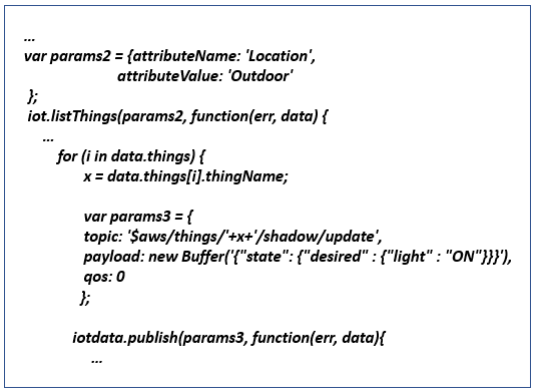


Figure 4.6: Lambda Function with ABAC Policy (Code Snippet)

to utilize the attributes of target resources on which IoT operations are performed. Suppose, a user wants *Sensor_1* to be able to publish data only on those lights which have an attribute *Location = {Outdoor}*. Currently, the AWS-IoTAC model cannot incorporate the attributes of target things/devices in IoT policies. This scenario, however, can be realized employing rules and lambda functions as illustrated in the following. The code snippet of the lambda function is presented in Figure 4.6. Here, first a search for things that have an attribute key and value is performed (e.g., *Location = Outdoor*), and a list of such things is obtained, i.e., *Light_1* and *Light_2* in this use case. Once the list is obtained, a message (in JSON format) to turn on the lights is published to their shadow update topic, as shown in Figure 4.6. The physical light bulbs receive the update message and change their states. As soon as the device state changes, a text message notification is sent to a user specified in the rules, *Rule_2* and *Rule_3*, through the AWS Simple Notification Service (SNS).

4.3 ABAC Enhancements to the AWS-IoTAC Model

This section discusses the available attributes in AWS IoT and their nature compared to real ABAC attributes. It then proposes the attributes required in AWS IoT to support a full form of ABAC policy and proposes some ABAC enhancements to the AWS-IoTAC model for enhancing the access control flexibility in AWS IoT. In a typical ABAC model, attributes of the users (actors), who are requesting access, and attributes of the resources (target objects), on which accesses are per-

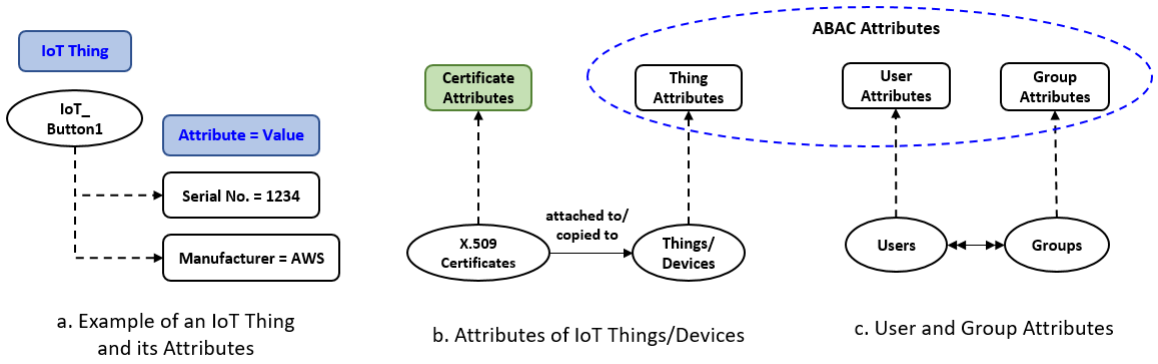


Figure 4.7: Attributes in AWS IoT

formed, are utilized in the access control policies to determine allowed permissions on objects. The attributes in ABAC are name-value pairs and represent characteristics of entities, such as users and objects. Often environment or system attributes are also brought into consideration. In AWS IoT, things can have a set of attributes. The attributes are defined for virtual things in the cloud and are synchronized with their associated physical devices.

4.3.1 Attributes in AWS IoT

An example of thing attributes is shown in Figure 4.7(a). Another way a thing can get attributes is through the certificate attachment or association as shown in Figure 4.7(b). Some attributes are set and defined while creating an X.509 certificate, and when a certificate is attached to a thing, then the attributes of this certificate can be used in AWS IoT policies to assign permissions to the things. However, a certificate attribute does not reflect any direct properties of the thing it is attached to and is thereby different than typical ABAC attributes.

Therefore, the access control model of AWS IoT (AWS-IoTAC) can be categorized as a restricted form of an ABAC model, mainly due to the following reasons.

- In AWS-IoTAC model, the attributes of only those IoT things/devices can be utilized which are requesting to perform actions on IoT resources (other IoT devices or applications) in the cloud.

- The thing attributes are applied in the policy only if the things/devices are using MQTT

protocol to connect and communicate to the AWS IoT service.

- In AWS IoT, currently, a thing can have only fifty attributes, of which only three are directly searchable.

4.3.2 ABAC Enhancements for AWS-IoTAC

Based on the above discussion and detailed exploration of the AWS IoT service, this dissertation proposes some enhancements for the AWS-IoTAC model in order to incorporate a more complete form of ABAC in the model.

1. **ABAC Including Attributes of Target Resources**

   As discussed in the use case scenario 2, the AWS-IoTAC model should be able to incorporate attributes of things/devices performing IoT operations as well as attributes of things/devices on which the operations are being performed, independent of the connection and communication protocol being used. The target resource attributes are mainly useful in isolating the identity of specific IoT objects. For example, an IoT device needs to publish messages to other devices which have some specific attribute values. The publishing device need not be aware of the particular topics it needs to publish to and can publish to multiple topics meeting some specific criteria, such as conditions in ABAC policies. Similarly, the policies should be able to incorporate attributes of both the source things and target things instead of specifying each resource (e.g., MQTT topics) in the policy.

2. **ABAC Including User and Group Attributes**

   A complete form of ABAC would require the inclusion of attributes of users and groups of users, as shown in 4.7(c). In real-world IoT systems, multiple users are using and controlling IoT devices. Therefore, including users and devices relationships through their attributes in access control decisions facilitates fine-grained authorization in CE-IoT platforms.

3. **Policy Management Utilizing the Policy Machine**

   AWS offers a form of policy-based access control based on policy files attached to entities

such as users, groups, "roles", and certificates. For all these entities, there are numerous policies defined. With billions of devices and their users, the policies for them will scale tremendously and soon become unmanageable. In the near future, a possible problem that AWS might encounter is a policy-explosion problem. While setting up the use case as an administrator, the need for a customer-based policy management tool is realized. Policy Machine (PM) [51, 52], an access control specification and enforcement tool developed by National Institute of Standards and Technology (NIST), could be utilized in this context. However, more detailed analysis of the AWS-IoTAC model with respect to PM would be required to demonstrate its applicability and feasibility in real-world scenarios.

4.4 Related Work

There has been significant research in IoT access control models, as recently surveyed by Ouaddah et al. [84]. Many of these models are based on capability-based access control (CAPBAC) [61] and role-based access control (RBAC) [54, 93], while there are a few utilizing attribute-based access control (ABAC) [63, 67]. In [59], a centralized CAPBAC model has been proposed based on a centralized Policy Decision Point (PDP). Whereas, a fully decentralized CAPBAC model for IoT is presented in [61]. However, a fully centralized or a fully decentralized approach may not be appropriate for managing the access control needs in a dynamic IoT architecture. Mahalle et al. [78] proposed an identity establishment and capability-based access control scheme for authentication and access control in IoT. Besides CAPBAC, an RBAC model is used for IoT in [77] where a thing's accesses are determined based on its roles. Similarly, Zhang and Tian [114] proposed an extended role-based access control model for IoT where access is granted based on the context information collected from system and user environment. These RBAC models for IoT still suffer from RBAC's limitations, such as role-explosion [89].

A hybrid access control model (ARBHAC) based on RBAC and ABAC is proposed by Sun et al. [71] to handle a large number of dynamic users in IoT. Here attributes are used to make user-role assignments, and then a user's roles determine accesses on resources or things. This ap-

proach is similar to *dynamic roles* [34, 74], where roles are dynamically assigned to users based on their attributes. However, ARBHAC lacks utilization of user, thing, environment and application attributes available in more general ABAC models.

The AWS-IoTAC model significantly differs from the existing models discussed above, especially in its nature of being an access control model developed for a real-world CE-IoT platform that is managed by the largest cloud service provider, AWS. Another distinguishing feature of this work is to identify the applicability of user attributes and attributes of IoT things (things/devices requesting access to other things/devices, and things/devices on which the access is being requested) in IoT access control. ABAC is a promising approach to address flexible and fine-grained access control requirements of the rapidly evolving IoT arena.

CHAPTER 5: ENHANCED ACO ARCHITECTURE FOR CLOUD-ENABLED INTERNET OF THINGS (CE-IOT)

This chapter enhances the recently published ACO architecture by Alsheri and Sandhu [35] motivated by a specific IoT domain, the Wearable Internet of Things (WIoT). Based on the Enhanced ACO (EACO) architecture, this dissertation develops an Access Control (AC) framework for the Cloud-Enabled Internet of Things (CE-IoT) to capture different types of accesses and communications within and between the layers of EACO architecture. The primary objective of the AC framework is to present a characterization of different kinds of interactions in CE-IoT which in turn will facilitate in the development of unifying standard access control models focused on specific interactions in CE-IoT. A remote health monitoring use case in the context of WIoT is also presented to demonstrate various interactions occurring in CE-IoT along with possible enforcement in a commercial CE-IoT platform, viz., AWS IoT. Finally, it discusses the objectives of the AC framework and relevant research directions. Major portions of text in this chapter are based on the following publication [41] with some revisions and modifications.

- Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. An Access Control Framework for Cloud-Enabled Wearable Internet of Things. In *3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 328-338. IEEE, 2017.

5.1 Internet of Things – Devices and Application Domains

Internet of Things (IoT) has given rise to a new wave of technology innovation. It has become a pervasive and diverse concept in recent years. IoT is an inclusive term in today's context which includes various enabling technologies: machine-to-machine (M2M) technologies, Internet, networking, communication protocols, cloud and mobile computing, and big data analytics [33]. A recent IoT architecture shaping the industry today is the combination of Cloud and IoT, with major cloud services providers offering IoT services and applications on top of their existing cloud

services [16]. The integration of Cloud and IoT has also been studied in the academic literature [45, 85, 90].

The Cloud-Enabled IoT (CE-IoT) architecture has gained much popularity in industry and academia recently. Most of the work is focused on either specific applications and technologies of IoT or state-of-art surveys. This dissertation focuses on access control aspects, mainly authorization and interactions, of the CE-IoT architecture. Securing the CE-IoT architecture involves security in two vast arenas—*Cloud* and *IoT*. A proper characterization of access control requirements in the CE-IoT architecture is necessary for its wide adoption and continued success.

Meanwhile, many commercial organizations are employing lucrative business models exploiting IoT and the significant amount of data associated with it. The IoT architectures being used in these businesses are driven by individual enterprise's needs and requirements rather than by some defined standard. On the one hand, it provides flexibility for interested entities to customize their IoT architecture, however, on the other hand, there is no consensus on a specific IoT framework. Alshehri and Sandhu [35] have developed an architecture for CE-IoT in general and named it as the Access Control Oriented (ACO) architecture. This architecture has addressed the inevitable need of the Cloud in the IoT and has introduced a *Virtual Object Layer* in the architecture.

The broad range of IoT applications and services has given rise to many sub-fields in the IoT space. Wearable technology, with its particular set of characteristics and application domains, has formed a rapidly growing sub-field of IoT, viz., Wearable Internet of Things (WIoT). While numerous wearable devices are available in the market today, aptly addressing the security and privacy concerns in WIoT are vital factors for wide adoption. Wearable devices are resource constrained by nature with limited storage, power, and computation. A CE-IoT architecture, a dominant paradigm currently shaping the industry and suggested by many researchers, needs to be adopted for WIoT. However, to capture different types of IoT objects and relevant components in WIoT, it is necessary to enhance the ACO architecture.

This dissertation enhances the ACO architecture from the perspective of WIoT by adding an **Object Abstraction Layer** to the four-layer architecture and then develops an Access Control

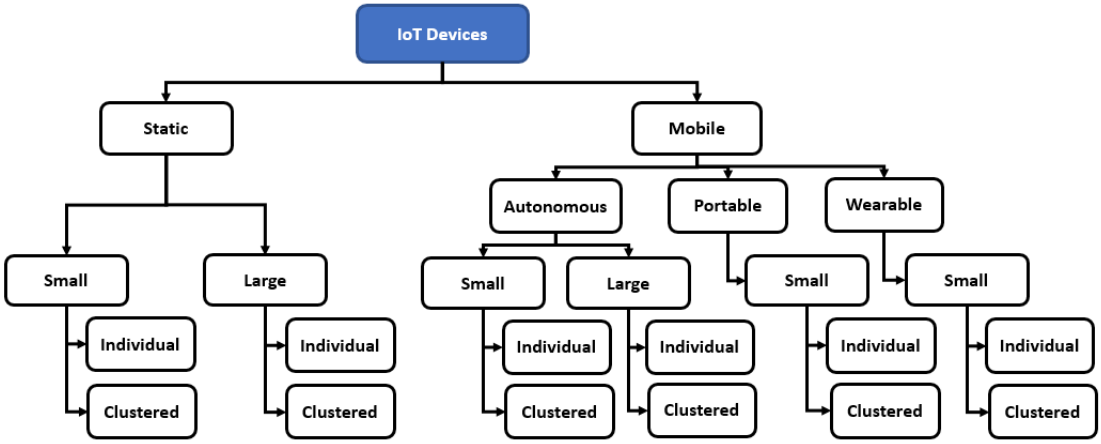


Figure 5.1: A General Classification of IoT Devices

(AC) framework to comprehensively represent various interactions between different layers of this enhanced ACO architecture. It also presents a general classification and taxonomy of IoT devices, along with a brief introduction to various application domains of IoT and WIoT. It then presents a remote health and fitness monitoring use case to illustrate different access control aspects of the AC framework and outlines its probable enforcement the AWS IoT service. Lastly, it discusses the objectives of the access control framework with some open problems.

5.1.1 A General Classification of IoT Devices

This section presents a classification of IoT devices based on three key characteristics: *mobility, size,* and *nature* of IoT devices. The classification and taxonomy of smart devices can be used to represent various sub-fields of IoT, such as Wearable IoT (WIoT) and Vehicular IoT (VIoT). It also discusses some IoT and WIoT application domains.

The impact of IoT is apparently visible in every aspect of human lives, such as smart homes, smart cities, offices, hospitals, and businesses. With the disruptive trend of IoT, different types of smart devices are evolving in the market with “anything” and “everything” being connected to the Internet. This widening IoT paradigm and increasing number of connected things requires a proper categorization of IoT devices/things. This categorization provides a holistic view of different types of IoT devices in the market today and can be extended as they evolve with time.

Figure 5.1 shows a general classification and taxonomy of IoT devices. To develop this classification, three main characteristics: *mobility, size,* and *nature* of smart IoT things are considered. They are defined as follows.

- **Mobility:** In IoT, the devices inherit the properties of their owners or of the entities to which they are attached. Mobility is one of the main characteristics that enables identifying the state of smart things and their capability of movement. This research classifies devices into two categories: **static** and **mobile**. Static things cannot move and are restricted to a specific location of installation, for example, a smart surveillance camera on a building. Whereas mobile things are capable of movement, and mobility can be achieved either independently (*e.g., autonomous cars)*, or dependently *(e.g., wearable smart watches)* through the device owners/carriers. Thus, they can be further classified into three categories: **autonomous** which are capable of moving independently or with the help of some external agent, **portable** which can be carried around, and **wearable** which can be worn and attached to their owners.

- **Size:** IoT devices are of different sizes, from a small tiny sensor to big complex machinery. It is difficult to define definite metrics to categorize IoT things based on the size. However, for simplicity here, two categories: **small** and **large** are considered. For example, any device that can be easily carried by an individual is a small IoT device, such as small sensors or wearable devices. Thus, only the *small* category is shown under *portable* and *wearable*.

- **Nature:** The third characteristic is the nature of things or devices. The nature of IoT devices depends on their architecture and functionality. Any thing that acts individually to perform a task is an **individual** IoT device, and a combination of multiple things that operate together to achieve a specific functionality is a **clustered** IoT device. As the name implies, individual things are made up of a single thing (e.g., a sensor sensing motion), and a clustered device is a combination of small sensors, such as wireless sensor networks (WSNs) or a smart car that has multiple sensors and actuators.

Other characteristics, such as technologies used, operating systems, network and communica-

tion protocols, can be considered for further enhancing the above classification as required. There have been other efforts to classify IoT devices. In [73], IoT things are classified into three categories based on the technology areas—*i) attached devices* (e.g., RFID[1] tags and barcodes attached to things), *ii) sensing and actuating devices*, and *iii) embedded devices* that have embedded processors and storage.

Another way of classifying IoT devices is based on communication capabilities resulting in two categories: *gateway devices* and *constrained devices*, where constrained devices are further classified into three classes: *Class 0, Class 1,* and *Class 2* based on their memory and processing capabilities [19]. In [70], the authors presented a classification of IoT devices for creating a security framework based on a comprehensive list of properties, such as power capability, real and non-real time, communication protocol, bandwidth, and size. Most of these works focus on distinct technologies used while classifying IoT devices and fall short of providing a general classification of IoT devices.

Based on various application domains, IoT has started to diverge into different IoT sub-fields, such as Vehicular IoT (VIoT), Medical IoT (MIoT), and Wearable IoT (WIoT). The objective of the IoT device classification presented here is to provide an overall general classification of heterogeneous IoT devices, and above three characteristics are believed to be the most suitable ones for this purpose. This categorization provides a basis to represent different IoT sub-fields, where distinct nodes in the tree can be combined to realize these sub-fields. For example, VIoT would be a combination of *autonomous, large,* and *clustered* IoT devices (sensors and actuators). Similarly, *wearable, small,* and *individual* or *clustered* device categorization can be realized as WIoT, as well as corresponds to MIoT to some extent. Therefore, this classification will enable IoT stakeholders, researchers, and businesses to focus on desired IoT sub-fields and associated security and privacy issues while developing innovative solutions.

[1]Radio Frequency Identification (RFID) allows automatic identification of things to which they are attached [109].

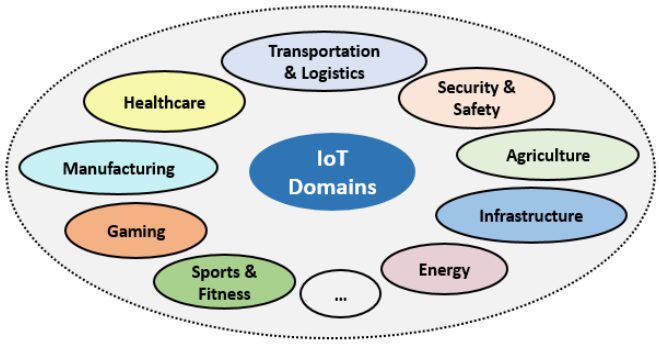


Figure 5.2: IoT Application Domains

5.1.2 IoT Application Domains

In recent years, numerous IoT services and applications are increasingly being deployed and explored practically in every domain, such as infrastructure, manufacturing, transportation, energy, as well as in critical domains like military and healthcare. In [37], five main IoT domains were presented along with their relevant scenarios. Since then IoT has influenced many other application domains as well and is still expanding.

Figure 5.2 presents some of the application domains being impacted by IoT today with the possibility of many more to be added. Smart cities, smart homes, and utilities are specific IoT examples in the infrastructure domain. Smart cities with IoT have been extensively studied [99, 104, 106, 113]. In transportation, RFID toll tags, smart traffic lights and traffic management with traffic flow data, parking with smart sensors, and mobile ticketing and travel are some IoT scenarios [37, 57]. Similarly, numerous IoT services and applications for healthcare have been proposed. A comprehensive survey by Islam et al. [66] discusses the state-of-art of IoT in health care, along with various medical IoT devices, services, applications, and use case scenarios.

Other domains like energy employ connected sensors for controlling and managing electricity usageand other forms of energy (e.g., wind energy, solar energy) for fulfilling the energy requirements of the planet efficiently. Meanwhile, retail and logistics are employing IoT for supply-chain management, moisture sensors are being used to assist in watering plants based on soil moisture and ultimately improve crop yields, and IoT devices and sensors are utilized to improve efficiency

and productivity in smart manufacturing [17]. Sports and fitness, security and safety, and gaming are some of the other emerging IoT domains, enabled by the wearable technology [29]. Besides these, the capabilities and benefits of IoT are being explored in many other domains, and soon enough will be realized in every aspect of our lives.

5.2 Wearable Internet of Things (WIoT)

Wearable Internet of Things (WIoT) is a rapidly emerging sub-field of IoT which has some distinct application domains. WIoT has already started to revolutionize the health care industry with numerous wearable devices and applications for monitoring vital body parameters, such as heart rate, pulse, temperature, blood pressure, blood sugar level, and other behavioral parameters [62]. Some of the examples of wearable devices, enabled by ubiquitous Internet and mobile technology, are smartwatches (e.g., Apple watch), fitness and health tracking devices (e.g., Fitbit), wearable health monitoring sensors (e.g., smart glucometer), and wearable smart clothing and accessories (e.g., smart t-shirts, necklace, bands).

Soon enough, WIoT will occupy one of largest market share among various IoT sub-fields. There has been significant research on wearable IoT devices and applications, mostly focusing on health care use cases. These studies are more driven towards the benefits and implementation of a particular application scenario. Since the wearable devices are directly associated to the users and collect their physical and behavioral data, user privacy and data confidentiality and integrity are fundamental issues impeding the success of WIoT. Therefore, the ACO architecture needs to be enhanced to address users' data and information security and privacy in CE-IoT.

5.2.1 WIoT Devices and Application Domains

Wearable devices are gaining popularity due to their light-weight nature and their capabilities of continuously tracking users for improving their quality of life. A wearable IoT device is defined as one that collects user data, processes and analyzes the data based on some intelligence, and delivers useful insights to the users [26]. In the context of the WIoT, currently, devices can be classified

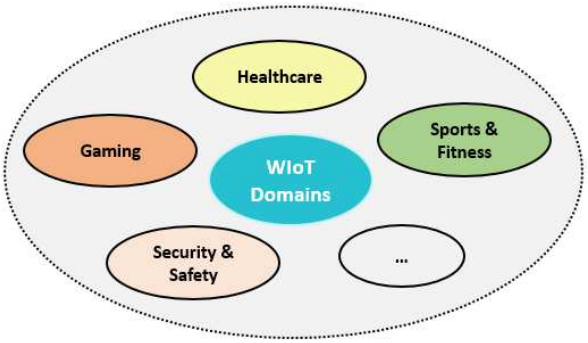


Figure 5.3: WIoT Application Domains

into three types: *In-Body, On-Body,* and *Around-Body*. **In-body** wearable devices are installed inside the human body, such as implantable devices (e.g., Pacemaker). **On-body** wearable devices are those which can be worn on the body, such as wearable sensors, clothing and accessories, and other contact-based sensors. **Around-body** devices are the ones which exist nearby the users. Generally, they work together with former two types of devices and gather data from users, such as user environment data, to perform defined functions. These devices on their own would not be considered wearable devices and are not true wearable devices. They are more like general IoT devices which work together with wearable devices, for example, a sensor collecting user environment data (e.g., location) that works with a wearable activity tracker device to perform some task. [29].

The application domains in this context are evolving, and innovative devices and applications are being introduced. Figure 5.3 shows the WIoT application domains which are discussed below.

Healthcare is one of the largest application domain of wearable devices today. IoT has many applications ranging from remote patient monitoring to assistance for chronic disease patients and elderly population. Wearable technology is the backbone of IoT in healthcare domain. With numerous wearable sensors and devices, the health of patients can be monitored remotely which helps in managing hospital resources. Wearable medical devices allow the patients to be more independent and be better aware of the benefits of a healthier lifestyle. Ambient assisted living is another application scenario of WIoT [50].

Similarly, in a busy world today, users desire to be active and are utilizing wearable devices for

maintaining fitness and a healthy lifestyle. In the realm of **sports**, it is essential for the players to track their performance, find their weaknesses and learn how to improve them for achieving their goals in life. Wearable devices for tracking activity and different body parameters, such as smart watches and bands, heart rate and pulse monitors, and pedometers, are available in the market today. The social media platform allows the users to share their data with other users [29].

One of the other important application domains is **security**. With numerous devices and platforms which a user needs to access, there is a possibility to incorporate user credentials in a wearable device. Nymi band [23] is one such wearable device that can be used as a multi-factor authentication device for a user who wears it to authenticate to different applications or services. Similarly, another unique application domain is **safety**. There are smartphone applications to track locations of different things and people. This capability can be easily extended to wearable devices that would track user location for safety purposes. For example, if there is a natural disaster and the location of users needs to be traced for rescue operations. Such devices are also beneficial to the people going on long treks or trips to dangerous places. Some of the wearable safety devices are discussed in [27].

Gaming is a booming industry where new advanced wearable devices are being designed for gamers. Virtual and Augmented reality enabled devices are developed to enhance user experience. Wearable gaming devices, such as head-mounted displays, will soon take over existing gaming devices [29]. WIoT has great potential in these domains, as well as other evolving domains. However, it needs strong authentication and access control mechanisms to support a huge architecture with billions of wearable devices and associated big data. A persistent problem in wearable technology is how to identify a user attached to a wearable device since the device could be intentionally or accidentally given to an unauthorized user. Authentication based on biometric parameters is an effective solution in such scenarios. Besides, authorization associated with these devices in a CE-IoT architecture is another critical issue that needs to addressed with further research.

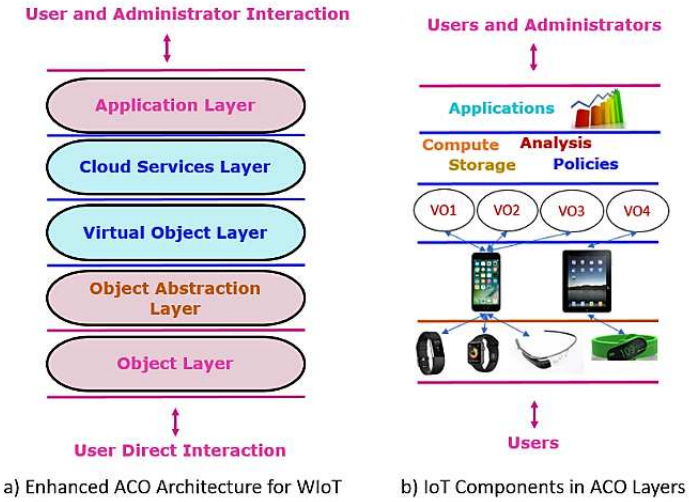


Figure 5.4: Enhanced ACO Architecture

5.3 Enhanced ACO (EACO) Architecture

Many different layered IoT architectures have been proposed in the literature [33, 37, 86, 110, 111]. In particular, an access control oriented (ACO) architecture for cloud-enabled IoT is proposed in [35], as presented in Chapter 2. The ACO architecture has four layers: *object layer, virtual object layer, cloud services layer,* and *applications layer*. Each of these layers encapsulates different entities, associated data, and their access control requirements in the CE-IoT framework. A detailed description of the ACO architecture has been presented in Chapter 2.

IoT devices, such as wearable devices, are usually resource constrained with limited computing, storage, and power. Therefore, these devices communicate to a unique device with comparatively better storage and computing power, known as *gateway devices*. These gateway devices abstract out the heterogeneity at the object level and facilitate pushing the data to a server or a cloud through the Internet. As mentioned earlier, due to the large amount of data generated by WIoT, a Cloud-enabled architecture is essential to support WIoT. This section enhances the ACO architecture [35] for IoT to incorporate different entities, components, and their associated communications motivated by a WIoT scenario.

This research considers the ACO architecture as the most relevant since it has been designed from an access control perspective and supports the CE-IoT framework. It has four layers: *object*

layer, virtual object layer, cloud services layer, and *application layer*. In general, these layers encompass all the aspects of CE-IoT. However, due to the heterogeneity and resource-constrained nature of wearable devices, there is a need for an abstraction layer which provides a gateway for the edge devices/things to communicate to the upper layers in the architecture. Therefore, the ACO architecture is extended by introducing an **Object Abstraction (OA) Layer** in the context of Cloud-Enabled WIoT.

Figure 5.4(a) shows the enhanced ACO (EACO) architecture. The OA layer is extended from the object layer and is comprised of gateway devices, such as smartphones. It has a unique task to facilitate object to VO communication abstracting all the heterogeneity (network and communication protocols) involved in the object layer. In the near future, when the edge devices become more sophisticated, the need for an abstraction layer may be reevaluated. However, the relevance of EACO architecture also exists beyond WIoT since many IoT devices, besides the wearable devices, are still resource-constrained with limited bandwidth and networking protocols (e.g., Low-energy Bluetooth). These devices require a more capable device, such as a gateway, to enable connection and communication with the Cloud infrastructure. In Figure 5.4(b), various components within each layer and their interactions are shown for a typical wearable IoT scenario with wearable edge devices, gateway devices, virtual objects, cloud services, and applications for monitoring and visualizing the IoT data.

5.4 Access Control (AC) Framework for EACO

Security and privacy in IoT are primary factors that will enable its broad adoption and continued success at the consumer level. Among the key technologies to achieve the objective of security and privacy are access control mechanisms. In general, access control requires both authentication and authorization techniques. However, this research focuses on the authorization mechanisms in a specific instance of IoT, the WIoT. In order to develop a comprehensive set of access control models for CE-IoT, an access control framework that captures different types of communications and data exchange within and among the five layers of the EACO architecture is necessary. The five

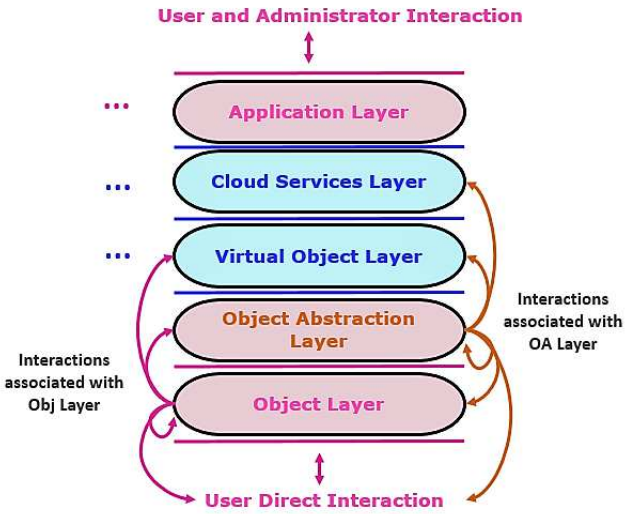


Figure 5.5: Interactions Between EACO Layers

layers of the EACO architecture encapsulate various entities, such as users, edge objects, gateway objects, virtual objects, cloud services, applications, and administrators, and these entities further comprise of other sub-entities. A single access control model would not be sufficient to capture all the access control requirements of different layers (and their associated entities) in the EACO architecture. Hence, this dissertation develops an Access Control (AC) framework for controlling accesses and communications (data exchange) between several entities in CE-IoT.

In the academic literature, many access control models have been proposed for IoT. Ouaddah et al. [84] extensively discuss access control models developed for IoT. The diverse and dynamic nature of IoT requires a unified access control framework for grouping different types of IoT models focusing on distinct IoT components and their interactions (access and communication). Figure 5.5 shows the possible interactions associated with two of the EACO layers (Object and OA) explicitly, where other layers' interactions follow the same pattern (represented as dots), in the five-layered EACO architecture. Here, it is assumed that each layer can interact with itself and its adjacent layers up to two levels in each direction (up and down). For instance, the interactions associated with the Object layer are: *i) with itself (Obj–Obj), ii) with users (Obj–Users), iii) with OA layer (Obj–OA),* and *iv) with VO layer (Obj–VO)*. There are numerous such interactions associated with each EACO layer where each one of them represents an access and authorization control point in

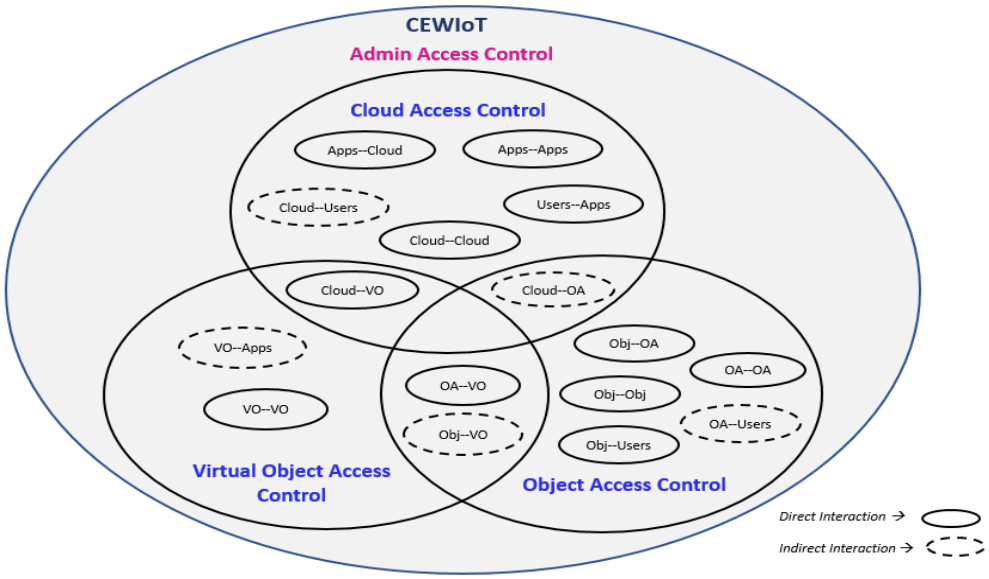


Figure 5.6: Access Control Framework based on Various Interactions in EACO Architecture

WIoT. The access control models addressing these control points are grouped into three categories of models: *i) Object Access Control, ii) Virtual Object Access Control,* and *iii) Cloud Access Control*, which comprises the Access Control (AC) framework for CE-IoT. Figure 5.6 depicts the AC framework incorporating all the possible interactions in EACO.

There are two modes of interaction between any two layers of the EACO architecture, first *direct interaction (DI)* and second *indirect interaction (IdI)*. For any layer, the DI implies interaction with itself and immediately adjacent layers; and IdI means interaction with the second level of adjacent layers at top and bottom of that layer. In the figure, DIs are shown as solid ovals and IdIs are shown as dashed ovals. There are some common interactions between any two category of models, such as *OA–VO*, and *Obj–VO* which belongs to both Object AC and Virtual Object AC models. This results in the overlap between the AC categories in the framework. The outer administrative access control circle in the framework represents that admin access control is relevant to the entire CE-IoT space, and administrative access control models can be designed for each one of the three AC categories. The interactions between layers of the EACO architecture are mediated by operational access control models, under configuration and control by administrators. The AC categories are discussed as follows.

- **Object Access Control Models:** This category of models includes the authorization at the Object layer and the Object Abstraction (OA) layer, as well as interactions with their adjacent layers (up to two level) in the EACO architecture. The edge IoT devices which are resource constrained reside at the Object layer, and gateway IoT devices that have sufficient resources for performing more substantial computation and storage functions reside in the OA layer. Access control models which focus on communications, and data access and transfer within and outside these layers can be grouped into this category of models. The interactions covered in this category are *Obj–Obj, Users–Obj, Obj–OA, Obj–VO, OA–OA, User–OA, OA–VO,* and *Cloud–OA*.

- **Virtual Object Access Control Models:** The access control models designed for virtual object (VO) communications among themselves (VO-to-VO), and for interactions with other layers can be grouped into the Virtual Object AC models. These models focus on interactions to and from the VOs and encompass three direct interactions *VO–VO, OA–VO, Cloud–VO*, and two indirect interactions *VO–Apps* and *Obj–VO*.

- **Cloud Access Control Models:** 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 [102]. The cloud is capable of hosting many IoT components. For example, AWS IoT hosts a device gateway, virtual objects, cloud services, and cloud applications. Thus, the access control models in this layer are more complex and may significantly overlap with above two categories. The interactions which need to be secured here are *Cloud–VO, Cloud–OA, Apps–Cloud, Users–Cloud, Cloud–Cloud, Users–Apps,* and *Apps–Apps*. The applications layer interactions are included within this category of models, since applications mainly utilize the data stored and analyzed in the Cloud to provide IoT services to the users. Also, these applications are often Cloud applications with their application and database servers hosted in the Cloud.

Any access control model developed for CE-IoT can be easily mapped to one of the above three

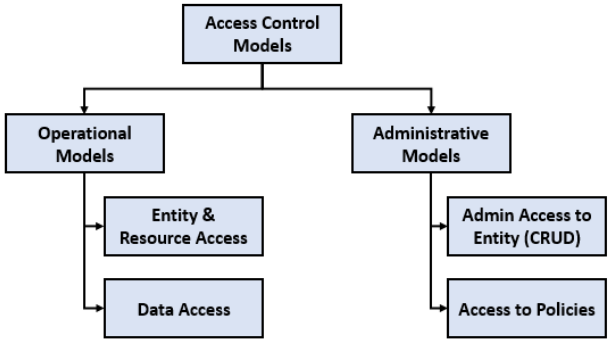


Figure 5.7: Types of Access Control Models

AC categories and may address authorization related to all the interactions (small circles inside a category) or a subset of the interactions relevant to that category.

5.4.1 Access Control Models

Access control models, in general, can be divided into two types: *Operational*, and *Administrative* models, as shown in Figure 5.7. An operational access control model secures usage of resources and services in any application or system. It also controls access to the data stored in a system. Administrative access control models control the access of admin users on resources and entities, such as create, read, update, and delete, and manage access to policies. Also, typically in any system, only the admin users have authority to specify and update the access control policies. As per the AC framework, each of the three categories of AC models for CE-IoT includes respective operational and administrative models. Role-based access control (RBAC) has been widely utilized in developing both operational and administrative models for various systems and applications.

In [84], Ouaddah et al. have presented a qualitative and quantitative analysis of access control models for IoT. As per their study, Capability-Based Access Control (CAPBAC) have been employed quite often for addressing authorizations in IoT. Moreover, other access control models, such as RBAC and ABAC, have also been considered. Each one of these has its advantages and disadvantages concerning the IoT domain. The benefits of CAPBAC are that it is user-driven and supports delegation; however, it does not consider contextual or environmental information in the

system. Whereas, ABAC employs contextual attributes (e.g., location, time, etc.), and user and subject attributes, and object attributes, but is often policy-driven rather than user-driven [84].

A Virtual Object AC model, addressing VO–VO communication, is developed by Alshehri and Sandhu in [36]. They developed operational and administrative access control models for controlling interactions between virtual objects (VOs). For operational models, they utilized access control lists (ACLs), CAPBAC, and ABAC, and for administrative models, they used ACLs and RBAC. Their work aligns with the AC framework presented here.

This dissertation developed an access control model for AWS Internet of Things, known as AWS-IoTAC [40], which has been discussed in Chapter 4. AWS IoT is a CE-IoT platform provided by Amazon Web Services (AWS) [1]. It controls the communications between several components, such as devices, virtual objects, cloud services, and applications based on the authorization policies defined for these entities in the Cloud. The AWS-IoTAC model was developed for a general CE-IoT platform and is an instance of ABAC to some extent, with policy-based access control as its core. This model fits into the Cloud AC model category of the AC framework and captures the interactions between cloud services and IoT entities.

Some of the suitable models to support the properties of CE-IoT in the context of WIoT will be influenced by ABAC and ReBAC, together with combining benefits of other models, such as CAPBAC. ABAC models are capable of incorporating the attributes of the users of wearable devices and relevant contextual attributes, such as the location of the users. The attributes of devices should also be considered in the access control model. Similarly, ReBAC models can be used to capture the relationship between users and objects in the context of wearable devices. For developing administrative models, RBAC provides great flexibility and administrative capabilities. ABAC is another suitable model for controlling admin authorization and functions. An administrative model for the hierarchical attribute-based model (HGABAC) [101] is developed in [58]. Influenced by such models, administrative models for operational WIoT models can be developed. These are some of our initial insights; however, more concrete access control models for CE-IoT concerning WIoT require further research.

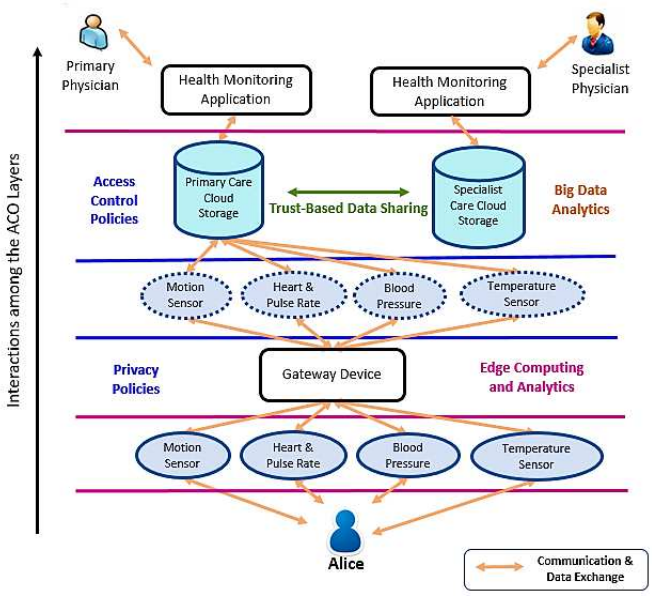


Figure 5.8: A Remote Health and Fitness Monitoring Use Case

5.5 Remote Health and Fitness Monitoring Use Case

Figure 5.8 shows a remote health and fitness monitoring (RHFM) example within the EACO architecture. This use case discusses the access control points along the EACO layers and how they map to the three categories of models in the AC framework. *Alice* has a problem of high blood pressure and uses wearable technology to monitor her health and overall fitness. At the Object layer, there are four wearable devices—*a motion sensor, a heart rate and pulse sensor, a blood pressure sensor,* and *a temperature sensor*, which Alice uses to measure relevant body parameters. These devices communicate to a gateway device (Alice's smartphone) at OA layer, which allows interaction with the upper layers of the architecture. OA layer provides an initial access control point where user-centric privacy policies can be deployed. It could also be used as a point to employ edge computing in the architecture. For each wearable device, there is a corresponding virtual object (*one-to-one association)* at the VO layer. VO layer facilitates seamless communication between applications and physical devices and addresses several IoT issues, such as identification, scalability, heterogeneity, security, and privacy [36].

The vast amount of IoT data collected by these devices is stored and analyzed in the cloud services layer. There are two data storages in the cloud, one for Alice's Primary physician and

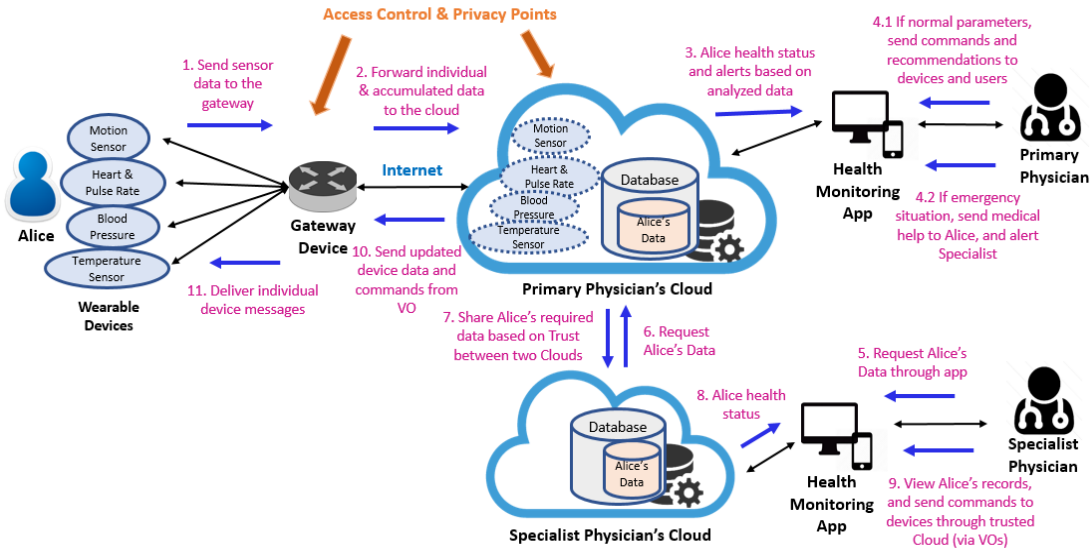


Figure 5.9: A Sequential View of RHFM Use Case

second for her Specialist physician. All the data is by default stored at her Primary physician's data storage. Alice's information is securely shared with the Specialist physician only when the need arises, for example when the Specialist or Alice request it to be shared, or in some emergency situation. Data security and privacy should be maintained based on trust established between the two physicians with user consent. Access control and privacy policies for any access control model, designed for either secure communications or data security and privacy, can be defined at the Cloud services layer as an Authorization service. The Cloud with ample resources also enables Big Data Analytics in WIoT. The analyzed data is then utilized by the Health Monitoring applications to show meaningful results to the physicians at the Application layer.

The interactions within and among different layers need to be authorized. For example, the edge wearable devices associated with a particular user must communicate with an authorized gateway device. Similarly, the gateway device must uniquely identify and authenticate the edge devices and allow authorized communication and data exchange with respective virtual objects. The access control models which would address such authorizations at the Object layer and OA layer and among their adjacent layers are a part of Object AC models. This use case assumes that the wearable devices do not communicate with each other. However, a possible scenario of communication between physical objects is WSNs, where each node can talk to every other

node in the network. Correspondingly, appropriate models addressing authorizations associated with virtual objects need to be developed. The Virtual Object AC models, as in [36], control access to virtual objects and relevant topics/channels in a publish/subscribe model. The Cloud AC models comprise models designed for controlling access within, and to and from cloud services and resources, as well as access control models developed for securing data in the cloud, and for enabling secure collaboration and data sharing between tenants, accounts, or multiple Clouds.

Figure 5.9 depicts a sequential representation of the use case. Alice uses four wearable devices—*a motion sensor, a heart rate and pulse sensor, a blood pressure sensor,* and *a temperature sensor*. The devices authenticate and communicate to a gateway device, which sends the collected data to their corresponding VOs instantiated in the Primary physician's Cloud. This data and information are stored in the database, and analysis is performed on it in the Cloud. The health monitoring application provides useful insights to the Primary physician based on Alice's data analytics results. If everything is normal, the Primary physician sends commands and recommendations to Alice by sending messages to the devices through the VOs.

Whereas in case of an emergency situation, immediate medical help is sent to Alice, and an alert (e.g., email or text message) is sent to the Specialist physician based on some predefined rules. Alice's data and analytic results are shared with the Specialist physician's cloud as required based on established trust and access control policies. The Specialist physician can also send commands to the edge devices, and schedule a visit for Alice and inform her through the application by posting updates to the device VOs in Primary physician's cloud. The gateway device ensures the delivery of messages sent by physicians to physical edge devices. In this scenario, there is no direct interaction between physicians or applications. The setup and configuration of the use case would be done by some administrators (cloud admin, health care admins, etc.) and the user (Alice).

5.5.1 Proposed Enforcement in AWS IoT

This section proposes an enforcement scheme for the above use case in the AWS IoT platform utilizing its services and functionalities. Previously, in Chapter 4, a smart-home use case is imple-

mented in AWS IoT, along with setting up its configurations and authorization policies for VOs, physical devices, and cloud services. In AWS IoT, a *Thing* for each wearable device needs to be created, which is its equivalent VO and has a *Thing Shadow* that provides a set of topics for clients (devices, apps) to publish/subscribe messages. For each device, a valid certificate registered in AWS IoT should be created and copied onto the physical devices. This is a complicated task since the devices need to be compliant with the AWS IoT protocols and standards.

AWS IoT has a device gateway which enables secure authentication and communication with edge devices. The idea of employing privacy-preserving policies defined by users or administrators at the gateway level requires further investigation since device gateway is embedded in the platform and cannot be accessed by the cloud users, probably due to security reasons. The IoT data generated can be stored in a *DynamoDB* database, and desired computation and analysis be performed utilizing AWS *Lambda function*. Based on the assumption that physicians use the AWS cloud, the application server for health monitoring applications would be hosted in the Cloud. However, in case of a collaborative data sharing scenario, appropriate cross-tenant or cross-account access control models for WIoT are currently missing in CE-IoT architecture.

5.6 Objectives of AC Framework

This section discusses the objectives of the AC framework for Cloud-Enabled IoT developed in the context of Wearable IoT and discusses relevant open research problems.

- **User-Based Device Authentication:** Wearable devices have peculiar characteristics of being closely related to their owners (who wear them, and whose information they are collecting). Therefore, physical security is of great importance, and also device authentication mechanisms based on user biometrics are necessary, such as fingerprint and heart rate. This ensures that even if a wearable device is lost or stolen, an attacker would not be able to comprise the data security and integrity. Such techniques for wearable devices are already being investigated [23], as well as need further research. The users should be able to remotely manage data stored in the devices, and at the same time, device firmware should be secure

enough (e.g., encryption technologies) to protect user data and information.

- **User-Centric Data Security and Privacy:** Wearable devices are attached to the users and collect very sensitive data and information that would compromise user privacy if it falls into an attacker's hand. Therefore, security practices involving the users are necessary for preserving data privacy and security in WIoT. It is crucial to include the users whose data is being collected in the authorization process, not at every step but at least at some initial point of the authorization and access control process. A recent study conducted on fitness tracker devices depicts threat to user's data due to vulnerabilities in the devices and provides guidelines for better security [9].

- **Edge Computing in WIoT:** Gateway device at OA layer is an ideal place to provide edge computing capabilities for constrained edge devices. One of the proposed mechanism of applying edge computing is cloudlets [96], which can be employed on the device gateways, such as a laptop or a small server machine at home. Edge computing is necessary for wearable devices due to their low bandwidth and low latency requirements which directly affects their usability. Edge computing in wearable cognitive assistance scenarios is discussed in [95]. For secure edge computing in WIoT, access control models for such scenarios demand significant research.

- **Multi-Cloud Architecture:** With more than 20 billion connected IoT devices by 2020 [20], the need for a multi-cloud architecture is inevitable to support IoT. A collaborative data sharing scenario across two Clouds is considered in above use case, yet appropriate Trust-based access control models for cross-tenant, cross-account, and multi-cloud architectures still lack in the context of WIoT.

CHAPTER 6: ATTRIBUTE-BASED COMMUNICATION CONTROL FOR CE-IOT

In the process of enhancing the ACO architecture and developing the Access Control Framework for CE-IoT, this dissertation identifies the need of communications control models, besides access control models, in the context of the CE-IoT architecture. In CE-IoT, IoT devices, gateways, VOs, and multiple Clouds are continuously communicating and sharing data with each other. It is critical to address security and privacy concerns associated with communication and data flow by developing secure and flexible communication control models. While access control models control access to objects by authorized subjects, communication control models are essential to secure communication and data flow from one component to the other and ensure user privacy in a system. There has been significant research on access control models; however, there is a lack of focused emphasis on communication control models in the academic literature.

This dissertation introduces a novel concept of Attribute-Based Communication Control (ABCC) to secure communications, and data and information flow, and to enforce user-driven privacy policies in any application or system. It develops a conceptual ABCC model and discusses its basic components and characteristics along with a comparison with respect to ABAC and its components. For securing communication and data flow in CE-IoT, this dissertation develops a formal ABCC model to control communications between the edge network and the Cloud infrastructure, known as the ABCC-EC model. This model is demonstrated through a Wearable IoT use case and a proof-of-concept implementation in AWS using its IoT service (AWS-IoT) and its edge computing service (AWS Greengrass). Finally, the performance evaluation of the proof-of-concept implementation of the ABCC-EC model is conducted to depict its applicability in a real-world use case scenario.

6.1 Attribute-Based Communication Control (ABCC)

Communication control has been widely studied in the networking domain. In networks, there are distinct devices and systems, such as routers and firewalls, which control communication occurring in the form of packets based on some predefined rules and algorithms. A more specific example of a communication control device or system in information security is a *Guard* [13]. Guards control communication from one component to the other in a network. While they have many similarities to firewalls, they are more secure in maintaining the confidentiality of the information and control the flow of information based on some defined constraints [13].

For example, a guard deployed between a *Top Secret* and a *Secret* network controls the flow of sensitive data and information from *Top Secret* to *Secret* level. Guards are secured trusted components mostly with trusted hardware and trusted software, hence ensure security and privacy even in scenarios where attackers try to compromise the network. They are also similar to a gateway to some extent in their functionality. The capabilities of guards in maintaining security and privacy in operating systems are studied and applied in [60].

The approaches and tools for securing communications depend on the communication model or architecture under consideration. Currently, one of most common communication paradigm adopted by many IoT platforms is the *Publish/Subscribe* model (e.g., MQTT *publish/subscribe*) which uses topics/channels for communicating messages between two components. One of the other popular communication paradigms is *TCP/IP* which allows communication between two systems. Similarly, another model is *Object-Oriented Programming (OOP)*, where objects communicate with each other by sending and receiving messages. Communication control models can be developed for and applied to any of these communication domains. In general, communication control has been studied in some domains (e.g., networks). However, communication control models have not been considered analogous to the access control models in the academic literature. Moreover, an attribute-based approach has not been applied in the context of communication control to the best of our knowledge.

In the Enhanced ACO (EACO) architecture for CE-IoT, developed in the previous chapter,

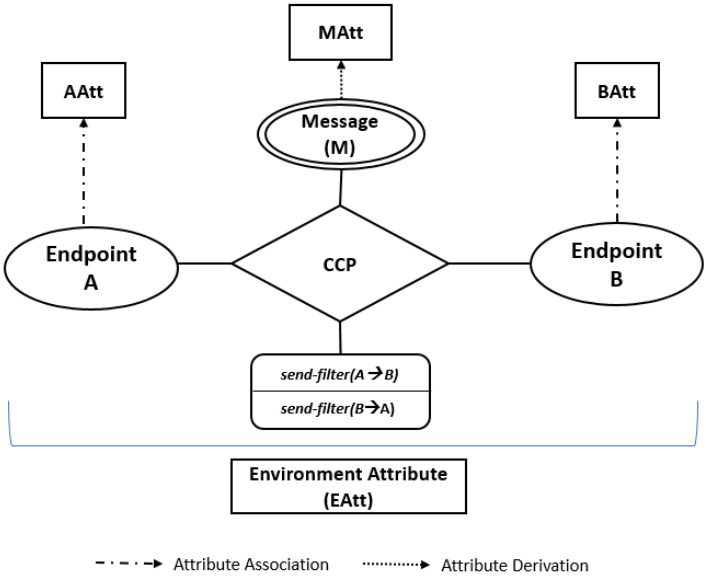


Figure 6.1: The Conceptual Attribute-Based Communication Control Model

there are five layers which capture various components of a widely dispersed IoT architecture. Within and between these EACO layers, there are different types of communications occurring continuously which need to be secured with appropriate communication control models. This dissertation proposes a novel concept of Attribute-Based Communication Control (ABCC). There have been some efforts in securing communication procedures using access control models [47,48]. Similarly, Alshehri and Sandhu [35] have identified the need to control data and communication in IoT and developed several models to control VO to VO communications using RBAC, CAPBAC, ABAC and ACLs [36]. However, a general conceptual model for ABCC is yet to be developed.

6.1.1 A Conceptual Model of ABCC

Figure 6.1 shows a conceptual model for ABCC. ABCC has unique characteristics compared to ABAC. There are two endpoints **EndpointA** and **EndpointB**, and a **Message** is being communicated between these two endpoints. The endpoints could be devices as routers, such as stateless/stateful routers and internal/external routers, or systems (computers), or IoT devices. As in an attribute-based approach, there are attributes assigned to *EndpointA* and *EndpointB* which represents the properties of these endpoints, such as *type, owner,* etc. The attributes of *EndpointA* and

EndpointB are represented as **AAtt** and **BAtt** respectively.

The *message* is a unique new element which is not a persistent object until an endpoint generates it or sends and receives it during the communication and data flow process. It is a structured message (e.g., JSON, XML) that comprises a set of properties. Thus, the message attributes and their values are derived from these properties within a message rather than being assigned by an administrator. **MAtt** represents the *message* attributes. The properties in the message content, which are in the form of key and value(s), can be derived as the message attributes. For example, if the message has a property as *temp = 80* where temp is the *key* and 80 is the *value*, then it can be derived as a message attribute *temp* with value *80*.

In ABCC, there is only one operation **send-filter**, but two instances of this operation depending upon the sender and the receiver of a message. The *send-filter*$(A \rightarrow B)$ represents a *send-filter* operation where the sender of a message is *EndpointA* and receiver is *EndpointB*. Similarly, *send-filter*$(B \rightarrow A)$ represents the communication from *EndpointB* to *EndpointA*, where *EndpointB* is the sender and *EndpointA* is the receiver. In order to secure communication and data flow, a set of communication control policies are defined by a user or an administrator based on the attributes of endpoints and messages in a system. For a specific sender, receiver, and a message, the **Communication Control Policy (CCP)** function is evaluated to identify if the message should be sent unfiltered (original message), filtered (removing sensitive information), or should not be sent from a sender to a receiver. As per the direction of communication, either of the endpoints can act as a sender or a receiver of a message.

Similar to ABAC, **Environment attributes (EAtt)** could also be included in *CCP* to enable more fine-grained and dynamic communication control based on respective context (e.g., time of day, location). A simple communication control policy is given as: *"if the owner of endpoint A and endpoint B is the same, then allow the message to be sent from A to B, otherwise deny."* The CCP function could be defined and be co-located with one of the two endpoints or hosted in a separate system between two endpoints. In CE-IoT architecture, the data and information is flowing between several components. For instance, in a wearable IoT scenario, IoT messages

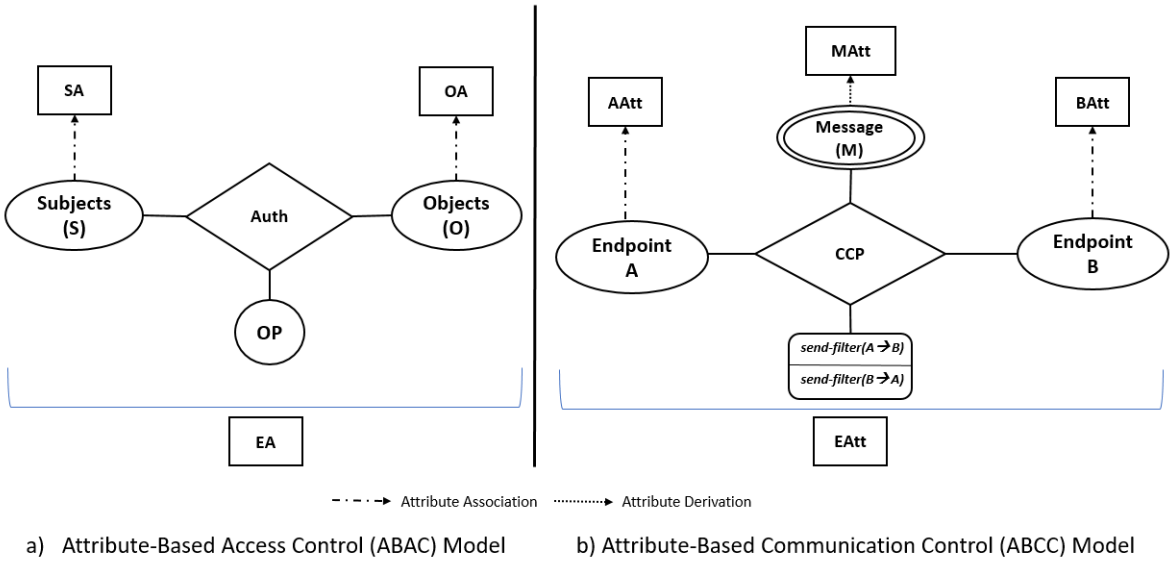


Figure 6.2: Attribute-Based Access Control vs. Attribute-Based Communication Control

are communicated between wearable devices, gateways, virtual objects (VOs), cloud services, and applications. Therefore, the endpoints, the messages, and the direction of communication and data flow will change according to the type of communication architecture under consideration.

This is a first general conceptual ABCC model presented for controlling communication between two endpoints based on their attributes as well as message attributes, to the best of our knowledge. This model is an abstract model and can be shaped into concrete entities and components as per the communication paradigm being used in a real scenario. Our proof-of-concept implementation presented later in the chapter will use the MQTT publish/subscribe model.

6.1.2 ABAC vs. ABCC

In general, access control refers to controlling access (e.g., read, write) to a protected entity (e.g., an object, or a subject) from another entity (e.g., a user or a subject) requesting that access on it. Whereas, in communication control, the communication of a specific element (e.g., message) is being controlled from one entity (or endpoint) to another. A conceptual ABAC model is shown in Figure 6.2 (a) and a conceptual ABCC model is shown in Figure 6.2 (b). In ABAC, attributes of different entities are used to determine allowed accesses on protected data and resources from

authorized entities requesting access on them. However, in Attribute-Based Communication Control (ABCC), the attributes of entities communicating with each other as well as the attributes of the communication unit, the message, both are taken into consideration while determining if the communication (data flow) should be allowed or not. While both of these models utilize attributes of various entities in the system, the units being controlled are distinctly different. In ABCC, the attributes of the communication unit are used together with attributes of other entities in the communication control policies. Another major difference is that ABAC protects data and information stored in the system which is static, whereas ABCC secures data and information in motion, such as data flowing from one entity to the other.

ABCC model is also distinct compared to ABAC since it is responsible for addressing two major security concerns. First, it identifies if two endpoints should be allowed to communicate with each other as per their attributes. Second, it controls the flow of data and information from one endpoint to another endpoint while considering the content of data and information. The basic components of ABAC are subjects, objects, operations and the authorization function, whereas the components of ABCC are endpoints, messages, the send-filter operation represented for each direction of communication, and the communication control policy. Moreover, in ABCC, the endpoints are system entities rather than individuals and represent machines in active states. A user's identity can be embedded in the attributes of the endpoints, and the message being communicated between these endpoints is associated with a specific user.

While ABCC and its capabilities are pertinent to many domains, this dissertation focuses on ABCC models in relevance to the CE-IoT architecture. It develops a formal ABCC model to control communications associated with the new *Object Abstraction layer* of the EACO architecture. In general, the specific elements and characteristics of the ABCC model depend on the system they are designed for and are managed by system administrators. For example, in a Linux operating system, access to objects, such as files and folders, is controlled by defining access control lists (ACLs) for each user in the system.

6.2 ABCC for Edge and Cloud Communication (ABCC-EC)

Internet of Things (IoT), a pervasive concept today, is being applied to every aspect of human lives from smart home, smart offices, connected cars, smart wearable, smart cities, and many more, enabling a smart world as a whole. For some IoT domains, especially where the devices are directly associated with the users, such as in Wearable Internet of Things (WIoT), the data being collected and analyzed by wearable IoT devices is highly sensitive. Moreover, a user's privacy is at risk at all times. For example, in a medical wearable IoT scenario where wearable devices are collecting a user's vital body parameters to monitor the user's health including analyzing this data to identify current or future health issues and perform some critical actions. Any unauthorized access to this data or an attacker compromising any of the components in the IoT system could result in a life-threatening situation.

With a tremendously growing number of smart devices and IoT applications, the concept of edge computing is widely explored in the realm of IoT architectures in both academia and industry. Concurrently, Cloud computing with a vast range of capabilities, such as storage, network, computation, and analytics, has become a popular paradigm to support the IoT infrastructure. In such IoT architecture where edge computing and Cloud computing intersect with each other, fine-grained and dynamic control on communication and data flow between different components, especially between the edge network and the Cloud, is necessary for user data security and privacy. This section develops a novel ABCC model to secure communications between the edge network and the Cloud in CE-IoT. This model is named as the ABCC-EC model. The real-world applicability of ABCC-EC is demonstrated through a proof-of-concept implementation of a wearable IoT use case in AWS IoT utilizing its edge computing service, AWS Greengrass [2], and AWS Lambda functions [4]. A performance evaluation is conducted to determine the feasibility of our implementation. Some performance enhancement techniques are also discussed and demonstrated.

6.2.1 ABCC-EC: Motivation

While Cloud Computing provides an attractive and cost-efficient framework for IoT, a single centralized Cloud-IoT architecture would not be sufficient. The number of connected IoT devices is increasing rapidly, and according to Gartner, there will be more than 20 billion connected devices by 2020 [20]. With so many connected devices, there is a vast amount of data associated and continuously being generated by them and not every bit of data is always useful or required to be sent or stored in the Cloud. Therefore, edge computing is necessary for effective IoT where some amount of computation, storage, and analytics capabilities are moved towards the edge of the network. Thus, an edge-centric approach in addition to the cloud-centric approach for IoT is necessary for its continued success in the future. In order to employ the concept of edge computing and distribute some of the cloud capabilities towards the edge of the network in CE-IoT, there is a need for multiple small edge clouds.

A new architectural component, cloudlets introduced by Satyanarayanan et al. [96, 97], is a mobility-enhanced small-scale cloud located at the edge of the Internet which extends today's cloud computing infrastructure [96]. The concept of cloudlets can be exploited to apply edge computing to existing cloud-centric IoT architectures. Using cloudlets in CE-IoT aids the goal of bringing computation and analytics closer to the edge of the network, where the IoT things reside. It provides a low-latency and high-bandwidth alternative to a high-latency and low-bandwidth end-to-end interaction between Cloud and edge IoT things.

• Edge Network of Things and Cloud Infrastructure

With the emergence of edge computing in CE-IoT, security and privacy of IoT communications and data have become a critical concern for IoT users and fine-grained communication control models need to be developed for CE-IoT. Figure 6.3 depicts a holistic view of the CE-IoT architecture mainly divided into two major components: **Edge Network of Things** and **Cloud**. The users, the smart objects and the gateways incorporating the edge virtual objects (EVOs) for each physical object together form an edge network, which is defined as the *Edge Network of Things (ENoT)*

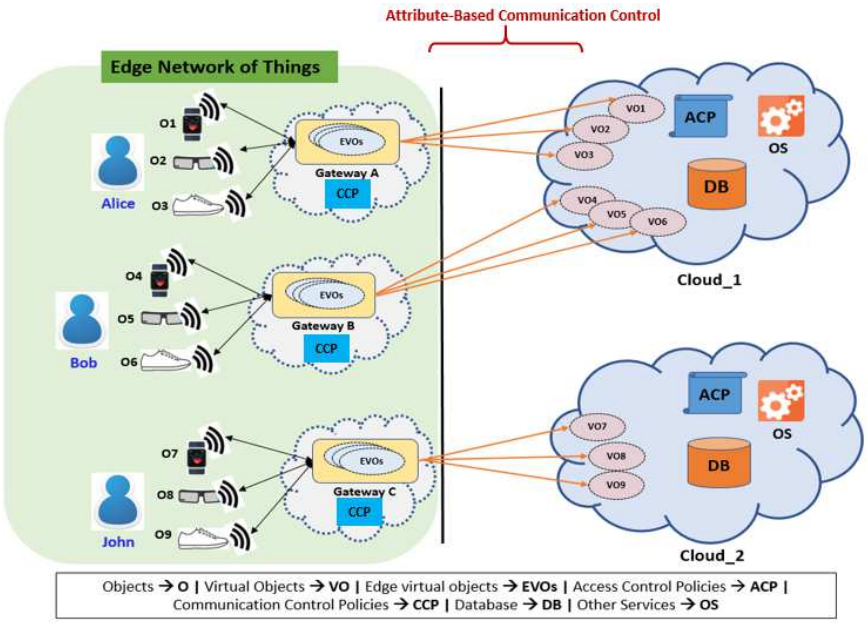


Figure 6.3: Edge Network of Things and Cloud

in this dissertation. The term *Network of Things* has been introduced by the National Institute of Standards and Technology (NIST) [107]. Here, the gateway also acts as an edge cloudlet [98] which enables edge computing, storage, and other capabilities similar to the Cloud but at a smaller scale. Being placed at the edge network, a gateway allows to employ communication control and privacy policies between the ENoT and the Cloud.

The ENoT space is assumed to be a trusted zone since the communication between IoT devices and gateway is secured utilizing X.509 certificates and public and private keys. Devices and gateways use the certificates and keys based mechanism for authentication purpose to the Cloud. For authorization, appropriate policy is attached to the certificate which is then installed on respective devices. These policies allow to send and receive messages between devices and gateways in ENoT. The messages are raw data being generated by the devices and sent to the gateway, or in other scenario, messages sent by the gateway to devices. However, the scope of this dissertation is limited to Edge and Cloud communication.

The *Cloud* incorporates the virtual objects (VOs) [82] (digital counterparts of the physical objects), access control policies (ACP) defined by the users and administrators, and databases and other services (OS) being utilized for analytic and visualization purposes in the IoT system.

The ENoT and Cloud architecture shown in Figure 6.3 is consistent with the EACO architecture shown in Figure 5.4 where the *Object Abstraction (OA) layer* separates the five layered architecture into two parts: *the edge network* and *the Cloud*. This architecture is an enhanced version of the Access Control Oriented (ACO) architecture proposed by Alshehri and Sandhu in [35] which is conformant with other previously published IoT architectures [33, 44, 57, 72, 85, 90]. A detailed description of the ACO architecture is presented in Chapter 2.

Currently, in most CE-IoT platforms, the smart devices send all the gathered data in the Cloud by default. This creates a set of vulnerabilities in the IoT network which can be exploited by attackers to execute attacks, such as eavesdropping, inference attack, etc. Therefore, communication control mechanisms for securing the IoT data plane, especially based on the properties of the data are inevitable in today's data-centric CE-IoT architecture. The two main objectives of this research are as follows.

1. **Minimize attack surface:** All the data being generated by smart devices is not always useful or required, and only the required data should be transferred to the Cloud. This minimizes the attack surface and allows to defend against some attacks. For example, an attacker is eavesdropping in the network to profile the user and gain her personal information (e.g., age group, gender, medical condition, etc.) based on its wearable devices data. Since the data being collected is selective data rather than continuous user data, it defends against such attacks in the network.

2. **Preserve user privacy:** User data is highly privacy-sensitive, such as personal information or behavior patterns, especially in a healthcare scenario. Generally, a user does not have control over the data being collected by smart devices. Users who would be concerned about their privacy and intend to protect their data outside of a trusted zone, i.e., their own ENoT, should be able to define and enforce user-centric privacy policies at the gateway level before forwarding it to Cloud for preserving their privacy. Moreover, controlling data flow from edge to Cloud will save a significant amount of network and Cloud bandwidth.

The primary goal here is to enable secure communication and data flow between ENoT to

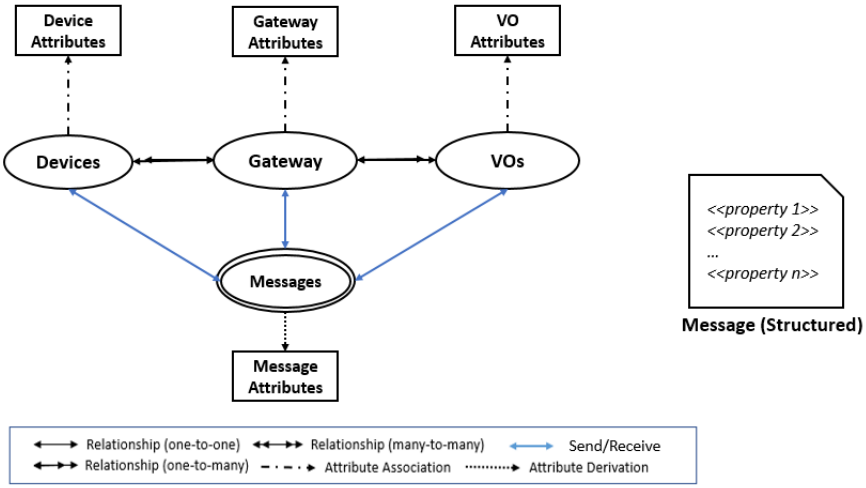


Figure 6.4: IoT Entities and Attributes in ABCC

Cloud, which in simple terms refers to control communication between gateways at edge network and VOs residing in Cloud as shown in Figure 6.3. For this purpose, we propose a novel Attribute-Based Communication Control (ABCC) model, the ABCC-EC model, to enable secure and fine-grained communication of IoT data and information between the edge network and the Cloud in a CE-IoT architecture. This model includes attributes of different entities (e.g., devices, gateways, virtual objects), as well as attributes of the communication unit being controlled, i.e., IoT data and information.

6.2.2 ABCC-EC: Model and Definitions

This section proposes and formally define an Attribute-Based Communication Control (ABCC) model to secure communications between the ENoT and the Cloud, known as the ABCC-EC model. In Figure 6.3, the communication between ENoT and Cloud occurs through the gateway and VO (in the Cloud). In terms of specific components, gateway devices, or simply gateway, in ENoT is an ideal place to enforce the ABCC model for CE-IoT. Thus, the communication control policies (CCP) are shown within the gateway cloudlet in Figure 6.3. In the process of developing the ABCC model, this section first discusses the entities and their relevant attributes of the model as follows.

• **Entities and Attributes**

As shown in Figure 6.3, there are two major components ENoT and the Cloud. The main goal here is to control data flow between ENoT and the Cloud, which is through their sub-components (*gateway* and *VOs*), respectively. As discussed earlier, a gateway is a more powerful IoT device which has comparatively better capabilities than the edge IoT devices. It can also act as a cloudlet by providing local compute and storage capabilities for the group of devices communicating with this gateway. It enables the communication and data sharing with the Cloud. The gateway has a set of attributes (with their values), such as *owner = {Alice}, type = {Gateway}, location = {Home}, etc.* These attributes represent the characteristics of the gateway.

With the separation of edge network and Cloud, the virtual objects should be identified in two separate ways, one as *Edge Virtual Objects (EVOs)* which are encapsulated in the gateway, and second as *Cloud Virtual Objects (VOs)* which are present in the Cloud as shown in Figure 6.3. In the rest of the chapter, Cloud VOs are simply referred to as VOs. The IoT devices communicate with the gateway which includes EVOs for these devices and in turn facilitates communication with their VOs in the Cloud. In Cloud, a persistent VO for each edge IoT device exists which represents the state (current and future desired) of a physical device. Any entity (e.g., devices, applications) communicating to edge devices through the Cloud will first send messages to the respective virtual objects which are later forwarded to the appropriate edge devices. There could be different types of association between the edge devices and the virtual objects as discussed in [35]. This research considers only the one-to-one object to virtual object association since devices (objects) and virtual objects are counterparts of each other in different environments. Hence, VO attributes are a subset of the device attributes. Some of the devices and VO attributes (and their values) are: *owner = {Alice}, type = {Device, VO}, location = {Home, Office, ...}, etc.* As gateway is also an IoT device, it might have some common set of attributes as devices, such as *owner, location, etc.*

Besides devices and gateway attributes, this research proposes that the IoT message (data or control message) being transferred from the devices to gateway and gateway to VOs also has some attributes which can be derived from the content of the message. As shown in Figure

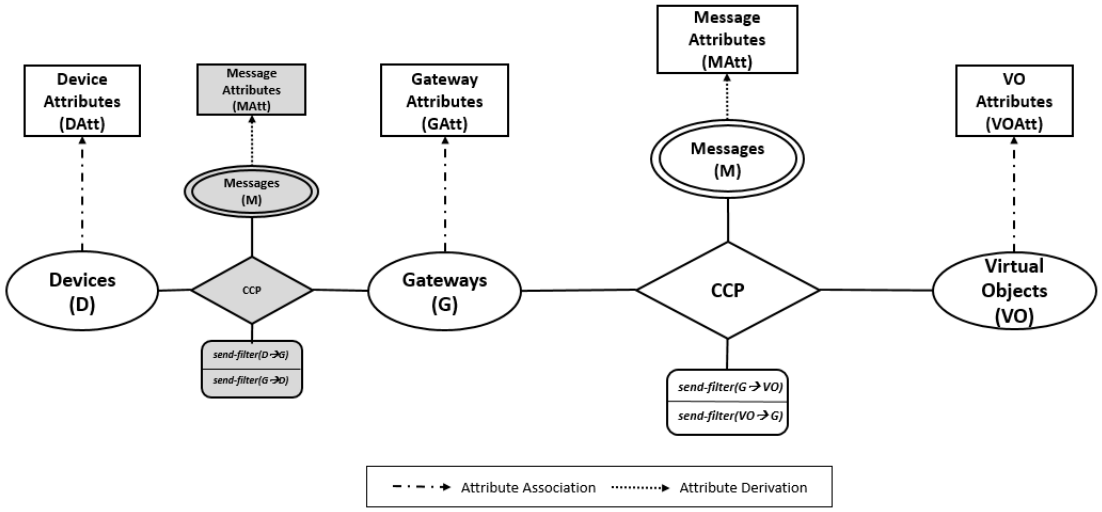


Figure 6.5: Attribute-Based Communication Control (ABCC-EC) Model for ENoT and Cloud Communication

6.4, *Message* comprises a set of properties, and each property consist of a *key* and *value(s)* (e.g., *heartrate = {75}*. The messages are sent or received by devices, gateways, and the VOs. The message attributes together with devices, gateway, VO, and environment attributes could be utilized while defining communication control and privacy control policies in the ABCC-EC model.

• **ABCC-EC Model**

Figure 6.5 presents the ABCC-EC model for controlling communication and data flow between ENoT and Cloud. The model is formally defined in Table 6.1. The entities of this model are **Devices (D), Gateways (G),** and **Virtual Objects (VO)**. *Devices* represent a set of IoT devices and *gateways* represent a set of gateway devices. *Virtual objects* are a set of virtual objects defined in the Cloud for each physical IoT device. Devices and gateways, and VOs and gateways have a many-to-one relationship, while devices and VOs have a one-to-one relationship. A **Message** is an IoT message (data message or control message) being communicated (sent/received) from one endpoint, a sender, to another endpoint (a receiver). It is the unit of control in the ABCC-EC model. A single message comprises a set of attribute and value pairs.

Devices, gateways, and virtual objects have a set of attributes, i.e., **Device Attributes (DAtt), Gateway Attributes (GAtt),** and **VO Attributes (VOAtt)**. In our ABCC-EC model, since de-

Table 6.1: ABCC-EC Model for ENoT and Cloud Communication

I. Core Components and Functions

- *D, G* and *VO* are finite sets of IoT devices, gateways, and virtual objects specified in Cloud respectively.
- *M* is a set of messages being communicated from one endpoint to another endpoint (gateways, VOs), where each $m \in M$ is the unit of control in any communication. A message *m* is a set of attribute value pairs.
- *DAtt, GAtt, VOAtt* and *MAtt* are finite set of device attribute, gateway attribute, VO attribute, and message attribute functions (referred simply as attributes) respectively.
- For each *att* in $DAtt \cup GAtt \cup VOAtt \cup MAtt$, Range(att) is a finite set of atomic values. The *time* is a special message attribute which is in the unbounded space.
- attType: $DAtt \cup GAtt \cup VOAtt \cup MAtt \rightarrow \{set, atomic\}$, specifies the types of the attributes as set-valued or atomic-valued.
- For each attribute att_d in DAtt, $att_d \colon D \rightarrow \begin{cases} Range(att_d) & \text{if } attType(att_d) = atomic \\ 2^{Range(att_d)} & \text{if } attType(att_d) = set \end{cases}$
- For each attribute att_g in GAtt, $att_g \colon G \rightarrow \begin{cases} Range(att_g) & \text{if } attType(att_g) = atomic \\ 2^{Range(att_g)} & \text{if } attType(att_g) = set \end{cases}$
- For each attribute att_{vo} in VOAtt, $att_{vo} \colon VO \rightarrow \begin{cases} Range(att_{vo}) & \text{if } attType(att_{vo}) = atomic \\ 2^{Range(att_{vo})} & \text{if } attType(att_{vo}) = set \end{cases}$
- For each attribute att_m in MAtt, $att_m \colon M \rightarrow \begin{cases} Range(att_m) & \text{if } attType(att_m) = atomic \\ 2^{Range(att_m)} & \text{if } attType(att_m) = set \end{cases}$
- *OP* is a *send-filter* operation that allows a sender to send filtered messages to a receiver as defined below.

II. Communication Control Policy

- *PLF* is a set of parameterized propositional logical formulas with formal parameters $s,\ r,\ m$ where $s \in G \wedge r \in VO$ or $s \in VO \wedge r \in G$ defined by the policy language in Table 6.2.
- A string $plf(s, r, m) \in PLF$ is evaluated for actual parameters $sen,\ rec,\ msg$ denoted as $plf(sen, rec, msg)$ by substituting attribute values of $sen,\ rec,\ msg$ in the logical formula and returns True or False.
- For a message $m = \{att_{m_1} = v_1, att_{m_2} = v_2, ..., att_{m_n} = v_n\}$, a filtered message on message attributes *MAtt'* is defined as: $m_{filter_{MAtt'}} = \{att_{m_i} = v_i \mid att_{m_i} = v_i \in m \wedge att_{m_i} \in MAtt'\}$ such that $MAtt' \subseteq MAtt$.
- **Communication Control Policy**:
  - $CCP_{send-filter} \subseteq PLF \times MAtt$, is a set of pairs each comprising of a string of a logical formula and a subset of message attributes.
  - A specific CCP defined as per the CCPL in Table 6.2 is evaluated using the following algorithm.

CCP_Evaluate$(s, r, m)\{$
  $m' = \phi$
  for each $(plf(s, r, m), MAtt') \in CCP$
    if $plf(s, r, m)$ *then*
      $m' = m' \cup m_{filter_{MAtt'}}$
  end$\}$

Table 6.2: Communication Control Policy Language (CCPL)

The communication control policy language (CCPL) consists of strings which are individually referred to as *plf(s, r, m)* that denote a parameterized propositional logical formula with formal parameters *s (a sender), r (a receiver), m (a message)* and return True or False, defined as per the following grammar.

- $\gamma ::= \gamma \vee \gamma \mid \gamma \wedge \gamma \mid (\gamma) \mid \neg\gamma \mid \exists x \in set.\gamma \mid \forall x \in set.\gamma \mid set \triangle set \mid atomic \in set \mid atomic \notin set \mid atomic \diamondsuit atomic$
- $\triangle ::= \subset \mid \subseteq \mid \nsubseteq \mid \cap \mid \cup$
- $\diamondsuit ::= < \mid = \mid \leq$
- $set ::= att_s(s) \mid att_r(r) \mid att_m(m)$, *for* $s \in G \cup VO,\ r \in G \cup VO,\ att_s \in GAtt \cup VOAtt,\ att_r \in GAtt \cup VOAtt, attType(att) = set$
- $atomic ::= att_s(s) \mid att_r(r) \mid att_m(m) \mid value$ *for* $s \in G \cup VO,\ r \in G \cup VO,\ att_s \in GAtt \cup VOAtt, att_r \in GAtt \cup VOAtt, attType(att) = atomic$

vices and VOs are the representation of physical IoT objects in physical and virtual environment respectively, the attributes defined for these entities are same. However, this may not be true in every scenario. An attribute is a function that takes an input (device, gateway, or virtual object) and gives a specific value or set of values as output from its range based on the type of attribute, i.e., atomic-valued or set-valued. The range of attributes is a finite set of atomic values. These attributes represent the characteristics of specific entities and are assigned by an administrator. Whereas, the **Message Attributes (MAtt)** are contained within the message and are derived based on its content. Each message is a set of attributes and their values. Among message attributes, *time* is a unique attribute which represents the time a message was sent or received, and its range is not finite since *time* is in the unbounded space. Here, we assume the properties of a message are known to an administrative entity (e.g., a user, a system administrator). The specific structure and properties of the message (e.g., JSON, XML) depends on the entities and the implementation platform.

The attributes are of two types: *set-valued* and *atomic-valued*. In *set-valued* attribute, a set of values are assigned to an attribute, while in an *atomic-valued* attribute, the attribute has a single value. There is only one operation **send-filter** with two instances of it representing the direction of communication *left-to-right* or *right-to-left*. The *send-filter* operation from the gateway to VO

is represented as *send-filter*($G \rightarrow VO$), and from VO to gateway as *send-filter*($VO \rightarrow G$). A set of parameterized propositional logical formulas is denoted as **PLF** where each *plf* $\in$ *PLF* is defined as per the *Communication Control Policy Language (CCPL)* given in Table 6.2.

A *plf* for a sender *s*, a receiver *r*, and a message *m* is denoted as *plf(s, r, m)* where $s \in G \wedge r \in VO$ or $s \in VO \wedge r \in G$ and $m \in M$, and is evaluated to True or False by substituting actual parameters for *s, r,* and *m* and their attribute values in the logical formula. For example, *plf(g:G, vo:VO, m:M)* $\equiv$ *gowner(g)* $=$ *owner(vo)* $\wedge$ *temp(m)* $>$ 102 where *gowner* $\in$ *GAtt*, *owner* $\in$ *VOAtt*, and *temp* $\in$ *MAtt*, is evaluated to be True if the attribute values for *gowner* of *g* and *owner* of *vo* are same and temp value for *m* is greater than 102 degrees Fahrenheit, else False. Correspondingly, a **message filter** for message *m* (a set of message attributes and values) on message attributes *MAtt'* is denoted as $m_{filter_{MAtt'}}$ and is a set of message attributes att_{m_i} and their values v_i such that $\{att_{m_i} = v_i\} \in m$ and $att_{m_i} \in$ *MAtt'*. For instance, consider a message generated by a NEST light bulb, *m = {color = Red, mode = On, manufacturer = NEST}*. For *MAtt' = {color, mode}*, $m_{filter_{MAtt'}}(m)$ *= {color = Red, mode = On}*, therefore, the filtered message *m'* contains the message attributes *color* and *mode* in *MAtt'* and their values. If *MAtt' = MAtt*, then $m_{filter_{MAtt'}}(m) = m$ which represents an unfiltered (original) message. If *MAtt'* $= \phi$, then $m_{filter_{MAtt'}}(m) = \phi$ which means all the message attributes are filtered and the filtered message is empty or null.

The **Communication Control Policy (CCP)** for the *send-filter* operation is a set of pairs of propositional logical formula string *plf* $\in$ *PLF* and a subset of message attributes and is denoted as $CCP_{send-filter}$. For a specific sender *s*, receiver *r*, and message *m*, the communication (and privacy) control policies are defined as per the policy language given in Table 6.2. It is evaluated as per the algorithm presented in part II of Table 6.1. For each policy tuple in CCP, the *CCP* function takes a sender *s*, a receiver *r*, and a message *m* as input and evaluates the *plf* included in CCP for *s, r* and *m* by substituting their attribute values in the logical formula. If *plf* is True, then the message *m* should be sent from a sender *s* to a receiver *r* with message attributes included in *MAtt'*. An example to depict the policy evaluation is as follows.

Suppose there is a user *Alice* who has a wearable IoT device with attributes *owner, type* that measures her heartrate and temperature. Thus, the message generated by the device has two message attributes *heartrate* and *temp* and their obtained values from Alice. This device communicates to a gateway *g* having attributes *gowner, type* which in turn communicates with the VO *vo* in Cloud that has attributes *owner, type*. Here, for controlling communication from the gateway to VO, there are two CCP policy tuples:

- $CCP_{send-filter(G \rightarrow VO)}$*(plf(g:G, vo:VO, m:M), MAtt')*, where
  plf(g,vo,m) $\equiv$ *gowner(g)* $=$ *owner(vo)* $\wedge$ *heartrate* > 105, and *MAtt'* $= \{$*heartrate*$\}$.

- $CCP_{send-filter(G \rightarrow VO)}$*(plf(g:G, vo:VO, m:M), MAtt')*, where
  plf(g,vo,m) $\equiv$ *gowner(g)* $=$ *owner(vo)* $\wedge$ *temp* > 102, and *MAtt'* $= \{$*temp*$\}$.

Here, for gateway *g*, VO *vo*, and message *m* = {*heartrate = 110*, *temp = 104*}, both tuples are satisfied and a union of message attributes in *MAtt'* is included in the filtered message *m'* along with their actual values, and is allowed to be sent from *g* to *vo*. A detailed use case for ABCC-EC is presented in the following section.

In the ENoT, there is a many-to-one association between the devices and the gateways. The data collected from the devices is sent to the edge virtual objects encapsulated in the associated gateway. Then, the gateway enables communication between devices and Cloud virtual objects (VOs). The device to gateway interaction occurs within the ENoT, or from an EACO architecture perspective, between the object layer and the OA layer and is considered to be outside the scope of this model. The ABCC-EC model considers that the communication between devices and gateways is secured through a tight coupling between them using X.509 certificates and keys, and authorization policies attached to these certificates. Therefore, the *CCP* function between devices and gateways is shown as grey color in Figure 6.5. In AWS IoT, there is a robust certificate and keys based coupling between devices and gateway. However, the *CCP* function between gateways and VOs secures the data flow and enforce user-centric privacy policies between edge network and Cloud. The ABCC-EC model is defined within a single Cloud architecture. However, the model could be extended

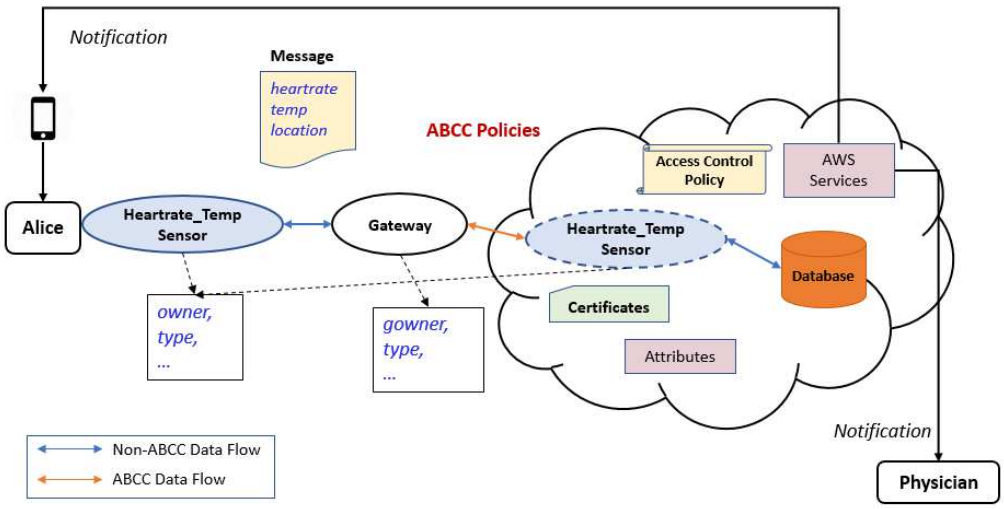


Figure 6.6: A Wearable IoT Use case in CE-IoT Architecture

to secure multiple gateways and multi-Cloud interactions. The enforcement of the ABCC-EC model in a CE-IoT platform is depicted through a remote health monitoring use case and a proof-of-concept implementation in AWS IoT using AWS Greengrass, which will be discussed in the following sections.

6.2.3 Use Case

This section illustrates the application of the ABCC-EC model within a wearable IoT use case in CE-IoT. Figure 6.6 depicts a *Remote Health Monitoring (RHM)* use case scenario. Within this use case, various components interact with each other by requesting access and exchanging messages to perform distinct tasks. Here, a user *Alice* and her physician both are monitoring her health data. The physician perspective is more detailed and focuses on Alice's health, existing conditions, and probable health risks or emergency situations, in case if critical conditions occur. Whereas, Alice wants to track her health and fitness in a general way. She prefers to be in charge of her data and is more concerned about her data privacy.

Alice has a wearable devices *Heartrate-Temp sensor* which senses her heartrate, temperature, and her location. This device is connected to a gateway device, shown as *Gateway* in Figure 6.6. The user, the device, and the gateway form an edge network. The edge network is connected to

a Cloud where a persistent representation of the physical IoT device, i.e., VO for *Heartrate-Temp sensor*, is created. The gateway receives messages (data or control messages) from *Heartrate-Temp sensor* and in turn sends them to its virtual object in the Cloud. The data is then stored in the databases and is processed/analyzed as required in the Cloud. For example, predictive analysis can be performed on Alice's data to find future risk areas, such as heart condition or stroke probability. This data could be used by specific applications which allow the user and the physician to track, visualize, and monitor Alice's health. Users and physicians can define some specific rules or conditions, and if these conditions occur, then they receive an alert or notification (e.g., text message, email).

In the Cloud, we created a VO for the *Heartrate-Temp sensor* as shown in the dashed oval. The wearable device and gateway must be authenticated and authorized to communicate with the Cloud and use its services. We implement this use case in the AWS IoT platform and utilize their certificate infrastructure (e.g., X.509 Certificate) and authorization policies for this purpose. X.509 certificates, keys, and access control policies authenticate device and gateway, enable secure communication between device and gateway, as well as allow gateway the permissions to use Cloud resources. Hence, the certificates and access control policies are shown inside the Cloud in the figure. Here, the gateway enables edge computing capabilities, such as local communication, storage, and computation. However, to overcome its constraints, it needs to communicate with a centralized Cloud and store required data and information in the Cloud and also use other Cloud capabilities. All the devices communicate with the gateway device which correspondingly sends and receives data and information to and from Cloud VOs.

In this scenario, only the authorized gateway should be allowed to exchange the device's data with specific VO in the Cloud. This is achieved by employing ABCC-EC policies which enable secure communication between gateway and VO and address user privacy based on the gateway, VO, and message attributes as in the ABCC-EC model. The physician monitoring the patient is only interested in tracking those values which are higher than some defined threshold value, for example, the physician want to specify a policy where heartrate and temperature sensor data flow

Table 6.3: WIoT Use Case in the ABCC-EC Model

I. Core Components and Functions
- DAtt = VOAtt = {*owner, type, wearable*}, GAtt = {*gowner, type*}, MAtt = {*heartrate, temp, location*}. - OP = {*send-filter*}. - U is a set of users in the system. - Range(*owner*)= Range(*gowner*)= U. - Range(*type*)= {*Thing, Gateway*}. - Range(*wearable*) = Boolean. - Range(*heartrate*) = {1 - 200}. - Range(*temp*) = {50 - 150}. - Range(*location*) = {*Home, Office, Other*}. - attType(*owner*)= attType(*gowner*)= attType(*type*)= attType(*wearable*) = attType(*heartrate*) = attType(*temp*) = attType(*location*) = atomic. - Thus, *owner* : $D \cup VO \rightarrow U$, *gowner* : $G \rightarrow U$, *type* : $D \cup VO \cup G \rightarrow$ *{Thing, Gateway}*, *wearable* : $D \cup VO \rightarrow$ *Boolean*, *heartrate* : $M \rightarrow$ *{1 - 200}*, *temp* : $M \rightarrow$ *{50 - 150}*, *location* : $M \rightarrow$ *{Home, Office, Other}*.
II. Communication Control Policy
- $CCP_{send-filter(G \rightarrow VO)}$*(plf(g:G, vo:VO, m:M), MAtt')*, where *plf(g,vo,m)* $\equiv$ *gowner(g)* $=$ *owner(vo)* $\wedge$ *heartrate* $\geq$ *110* $\wedge$ *temp* $\geq$ *102*, and *MAtt'* $= \{$*heartrate, temp, location*$\}$. - $CCP_{send-filter(G \rightarrow VO)}$*(plf(g:G, vo:VO, m:M), MAtt')*, where *plf(g,vo,m)* $\equiv$ *gowner(g)* $=$ *owner(vo)* $\wedge$ *heartrate* $<$ *110*, and *MAtt'* $= \{$*heartrate, temp*$\}$. - $CCP_{send-filter(G \rightarrow VO)}$*(plf(g:G, vo:VO, m:M), MAtt')*, where *plf(g,vo,m)* $\equiv$ *gowner(g)* $\neq$ *owner(vo)*, and *MAtt'* $= \phi$.

to the VO in Cloud only when the data matches the specified condition, e.g., "heartrate is greater than 80." Similarly, *Alice* is concerned about her privacy and does not wish to share her location information at all times. To enforce such policies, the message attributes in ABCC-EC can be employed while defining the communication control policies. Such communication and privacy control policies can be easily applied at the gateway level utilizing the ABCC-EC model.

Table 6.3 presents the use case scenario with a device, VO, and gateway attributes, and specified ABCC-EC communication control policies. This use case has been implemented utilizing AWS Cloud and its IoT, edge computing and other services. Here, the device (*Heartrate-Temp sensor*) and its VO has same attributes: *owner, type,* and *wearable* which represents their owner, type of device (thing or gateway), and if its a wearable device. Similarly, the gateway has *gowner* (gateway owner) and *type* attributes. The message has three attributes: *heartrate, temp,* and *lo-*

cation. The range and attribute type of the attributes are presented in part I of the table. The communication control policies are defined in part II and are evaluated for actual parameters and their attribute values to allow or deny data flow from the gateway to the VO. The first CCP represents a critical scenario where heartrate and temperature of the user are higher than the normal values. For instance, the user wants the location attribute in the message to be shared with the external network, i.e., Cloud, only in emergency situations. Thus, none of the message attributes are filtered in *MAtt'*, and unfiltered message *m* is sent from a gateway *g* to VO *vo*. In the second CCP, in a normal scenario when heartrate is normal, a user wants to filter the *location* attribute in the message flowing from edge to Cloud. Lastly, if the owner of *vo* and *g* is not the same, then the message should not be sent. Hence all the attributes in the message are filtered.

Other example scenarios could be when the user or the physician would not require to get updates from the devices (e.g., at a certain time or location, or if the physician is on vacation). Such scenarios can be defined formally and implemented in real-world platforms. A proof-of-concept implementation of the ABCC-EC model within the scope of the use case defined in Table 6.3 is presented in the next section. Besides, for any use case, the entities, relations, and authentication and authorization schemes depend on the application platform where it is being implemented and can be adapted accordingly.

6.2.4 Implementation

This section presents a proof-of-concept implementation of the ABCC-EC model in the context of the remote health monitoring use case presented in the Use Case section. It discusses the enforcement architecture and demonstrates the use case scenario defined in Table 6.3. The application platform used for the implementation is Amazon Web Services (AWS) including a variety of its services, mainly its IoT and edge computing services, viz. AWS IoT and AWS Greengrass.

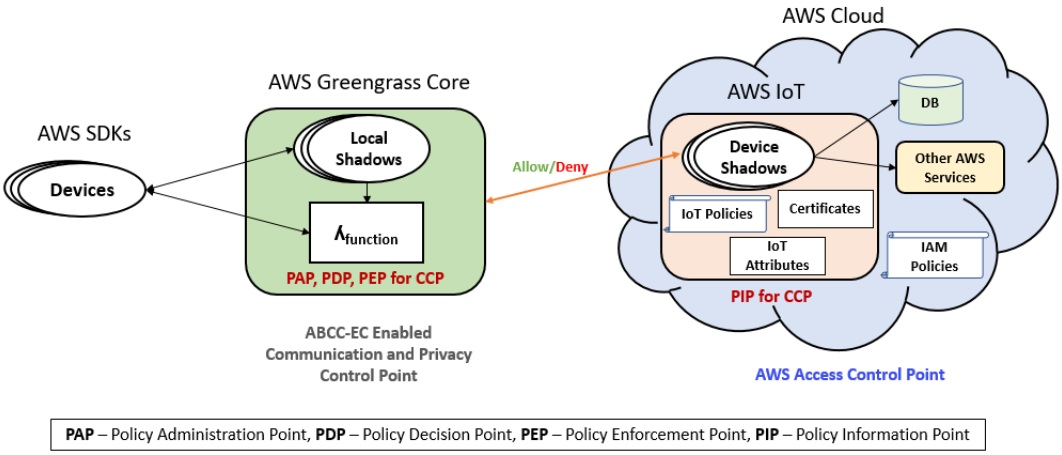


Figure 6.7: Implementation Architecture Utilizing AWS IoT and AWS Greengrass

- **Enforcement Architecture**

The enforcement architecture comprises AWS Cloud and its services which are utilized to develop different components of the use case and enforce the ABCC-EC model. Figure 6.7 shows the enforcement architecture for ABCC-EC utilizing the AWS Cloud, AWS IoT, and AWS Greengrass services. An active AWS account is required to set up the implementation architecture. The IoT device is simulated using the AWS SDKs, mainly AWS Python SDK [5] is used. For each device, a virtual thing is defined in AWS IoT using its web management console and with each thing is an associated thing shadow (or device shadow) which provides a set of MQTT topics (channels) to interact with the device shadow. The virtual things (Cloud VOs) in AWS IoT are represented as *Device Shadows* in Figure 6.7.

In this work, the focus is to secure communication and data flow from a gateway to VO. However, to address access control requirements and communication control between devices and gateways, we utilize the existing security mechanism in AWS. For each physical IoT device, there needs to be specific X.509 certificate created and attached to the thing in AWS IoT. Then, the certificate (with private key) is securely copied onto the physical device. The certificates are used for authentication purpose, while authorizations are managed by IoT policies defined based on the requirements and attached to a respective certificate to enforce it on the associated device. AWS

Figure 6.8: IoT Policy for Greengrass Core

IoT provides the capability to define attributes for IoT devices; however, only 50 attributes can be defined for each thing. The detailed description of the access control mechanisms in AWS IoT is discussed in Chapter 4 which also presents a formal AWS-IoT access control model, known as the AWS-IoTAC model [40].

A relatively new component of this architecture is the *AWS Greengrass Core*. AWS Greengrass is a new service recently introduced by AWS to provide local computation and communication capabilities for IoT devices towards the edge of the network. As per AWS, it is described as [30]: *"AWS Greengrass is software that extends AWS cloud capabilities to local devices, making it possible for them to collect and analyze data closer to the source of information, while also securely communicating with each other on local networks."*

The AWS Greengrass is a service in the AWS IoT. There is a concept of a Greengrass group in AWS which comprises a Greengrass core and a set of IoT devices associated with this group. The AWS Greengrass core is also an IoT device and has a thing and thing shadow defined for it. Therefore, we can define attributes for Greengrass core in AWS IoT. Similarly, a certificate and policies are also attached to the core device. An access control policy created by default for Greengrass core device during its set up in AWS is presented in Figure 6.8. It gives permission to the core device to perform all *iot* and *greengrass* actions on any resource in AWS. Currently, the MQTT communications associated with the Greengrass is enabled through a *subscription table* where a source, a target, and a MQTT topic is specified. It allows a source (e.g., device, Cloud,

Lambda) to forward MQTT messages on a specific topic to a target (e.g., device, Cloud, Lambda). However, it does not enable attribute-based control as well as does not include message content level control. In the context of this research, the AWS Greengrass core acts as the gateway device. It provides an interface between the edge devices and the Cloud infrastructure and includes edge computing capabilities similar to a cloudlet in the AWS CE-IoT architecture.

The local computation is enabled through another AWS service, i.e., AWS Lambda function [4]. The AWS Lambda service allows the users (or customers) to define customized programs as *Lambda functions* in the service which can be programmed (in supported languages Python, JAVA, Node.js) to achieve the desired functionality. It requires proper authorizations to be configured using IAM roles and policies so that these functions have necessary permissions to act on AWS services and resources [4]. The customized Lambda functions are deployed in the core device (gateway) to enforce ABCC-EC policies between the gateway and virtual things (VOs) in AWS IoT. Therefore, the AWS Greengrass core acts as the Policy Administration Point (PAP), Policy Decision Point (PDP), and Policy Enforcement Point (PEP) for the ABCC-EC model. Since the device information and attributes are defined in the Cloud, the AWS IoT acts as Policy Information Point (PIP) for communication control policies. The Greengrass core device is simulated as an AWS compute instance (EC2 virtual machine).

• **Enforcement of ABCC-EC Enabled Use Case**

This section discusses the enforcement details of the ABCC-EC use case presented in Table 6.3. It is enforced utilizing the architecture shown in Figure 6.7. Here, Alice owns the wearable *Heartrate-Temp Sensor* which continuously collects her heartrate, temperature, and location data. A *Heartrate-Temp Sensor* device is programmed in Python using AWS Python SDK, and a thing shadow (VO) for it is defined in AWS IoT which receives the device data through the gateway. The gateway is built as a Greengrass core device using Greengrass software. The *Heartrate-Temp Sensor* is connected to the core device and sends messages to it. The Greengrass core device receives the messages from the sensor and sends them to its Cloud VO. The ABCC-EC policies are

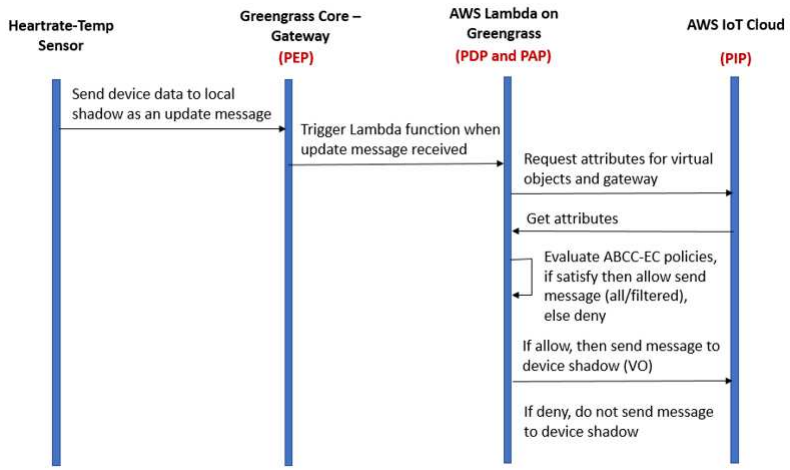


Figure 6.9: Sequence Diagram for ABCC-EC Policy Evaluation

specified in a Lambda function which is deployed in the Greengrass core.

We defined attributes for the sensor, its VO, and the core device in AWS IoT. Sensor and its VO attributes are *{owner, type, wearable}*. Here, the *owner* is an atomic attribute since only a single user can own a wearable device at a certain time, who is Alice here. The *type* represents the type of device *Thing* or *Gateway*, and *wearable* is an attribute that represents the boolean value, if the device is wearable, then the value is *True*, else *False*. The Greengrass core has attributes defined, which are *gowner* and *type*. The owner of the gateway is Alice, and the device type is Gateway. Similarly, the device messages has attributes *heartrate, temp*, and *location*. In AWS IoT, the messages (data or control message) are in JSON format, and different properties (attribute and value pairs) in the messages can be derived at the Greengrass core device when a message is received. Based on the attributes of the sensor, its VO, and the Greengrass core device, as well as the message attributes, the ABCC-EC policies are defined to control the flow of messages (device data) from Greengrass core (gateway) to device shadow (VO) in AWS IoT.

The first condition in the parameterized logical formula in CCP enforces that *"a gateway can communicate to a VO in Cloud only if the owner of both gateway and VO is same."* The second condition involves message attributes that can be derived from the message, i.e., *"if the heartrate is greater than or equal to a threshold value, which is 110, and if the body temperature is greater than or equal to 102."* Therefore, the message is sent with all the message attributes and their values

```
...
#get VO and gateway attributes and their values
def getAttributes():
        thingAttributes = iotClient.describe_thing(thingName='THINGNAME')
        thing_owner = thingAttributes['attributes']['owner']
        thing_type = thingAttributes['attributes']['type']
        gatewayAttributes = iotClient.describe_thing(thingName='GATEWAYNAME')
        gateway_owner = gatewayAttributes['attributes']['gowner']
        ...

getAttributes()

def function_handler(event, context):
        #getMessageAttributes and their values from event (IoT message)
        heartrate = event["state"]["desired"]["heartrate"]
        temp = event["state"]["desired"]["temp"]
        ...
        #evaluate ABCC policies with logical formulas for specific message attributes
        evaluateCCP()
                ...
                if thing_owner == gateway_owner and int(heartrate) > 110:
                ...
                iotDataClient.publish(topic='$aws/things/THINGNAME/shadow/update',
                payload="MESSAGE_WITH_SPECIFIC_ATTRIBUTES")
                ....
        return
```

Figure 6.10: Lambda Function with ABCC-EC Policy (Code Snippet)

from Greengrass core to *Heartrate-Temp Sensor* VO in AWS IoT. Similarly, the second CCP policy tuple enables a user to enforce privacy policies by removing some sensitive message attributes. The third policy tuple enforces a strict ABCC policy which states that if the owner of gateway and VO are not same, then no messages are sent from Greengrass core to sensor VO. Therefore, ABCC-EC policies defined on gateway and VO attributes along with message attributes enables to enforce fine-grained communication control and user-specific privacy policies at the granularity of the message content.

Figure 6.9 depicts a sequence of actions while evaluating the ABCC-EC policies. Whenever the Greengrass gateway receives a message from a device, a Lambda function with ABCC-EC policies is triggered which gets required attributes and their values and evaluates defined ABCC policies to allow or deny to send messages from Greengrass gateway to VO with specific message attributes. Figure 6.10 shows a code snippet of our Lambda function for this use case where attributes of involved entities are obtained and then specified ABCC policies are evaluated to make communication control decisions.

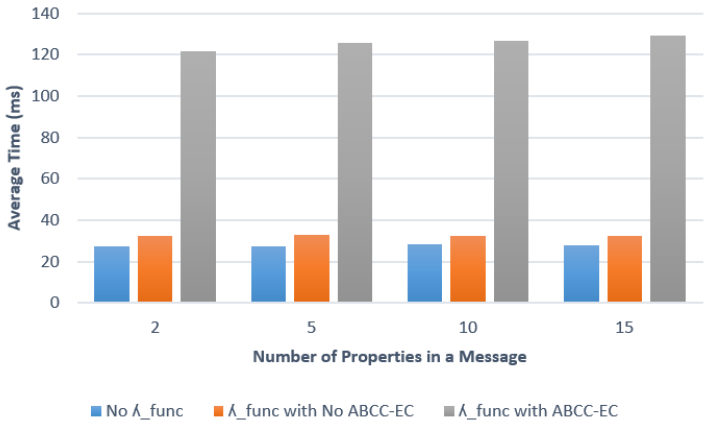


Figure 6.11: Device Shadow Update Time

6.2.5 Performance Evaluation

This section presents a performance evaluation conducted on our proof-of-concept implementation to enforce the ABCC-EC model in AWS IoT and AWS Greengrass. Here, the primary goal is to evaluate the overhead due to the addition of Lambda function and the ABCC policies specified and enforced through the Lambda function. In each set of experiments, the time when a message is sent by a device and received by the device shadow in the Cloud is recorded to calculate the time taken for edge device to cloud VO communication in different scenarios.

Figure 6.11 represents initial results. The time depicted in the y-axis is the average time in milliseconds for 20 message update requests sent by the device and received by the device shadow in AWS IoT. The x-axis represents the message size (i.e., number of properties in a message). The three bar charts represent different scenarios: first, without a Lambda function at the gateway and device shadow sync turned on; second, with a Lambda function deployed in the gateway but no ABCC-EC policies defined; and third, with a Lambda function deployment including specified ABCC-EC policies. Here, the average shadow update time is lowest when there is no Lambda function deployed at gateway as expected. The average shadow update time with a Lambda function not including ABCC-EC policies, though slightly higher, is quite similar to the case with no Lambda function. However, the average shadow update time with a Lambda function including ABCC-EC policies is significantly higher than the other two categories.

Figure 6.12: Device Shadow Update Time with Attribute Caching

One of the reasons identified for the significant difference between Lambda with no ABCC-EC policies and with ABCC-EC policies is the round-trip time between the gateway (Greengrass) and Cloud (AWS IoT) for getting attribute information. Each time a new message was received, and ABCC policies need to be evaluated the Lambda function request and get attributes for thing and gateway from AWS IoT. As a solution to this problem, we implemented attribute-caching in the Lambda function. Therefore, for the first message, the attributes are queried from the Cloud and then are cached at the Greengrass core. This significantly reduced the round-trip time, and except for the first message, for the rest of the messages, the policy is evaluated utilizing the cached attributes and values. A new set of experiments were executed for the Lambda function including ABCC-EC policies with attribute-caching capability. The attribute-caching capability enabled significant performance improvement in the results. Figure 6.12 shows the results with the performance enhancement. The graph in the figure depicts only a few milliseconds difference between the average time for Lambda without ABCC-EC and Lambda with ABCC-EC. In both the results, the size of the message, i.e., the number of properties (attributes) in the message, do not necessarily affect the performance since only some of these attributes were included in the ABCC-EC policies. However, in experiments with several message attributes included in the policies, there wasn't a considerable difference in average device shadow update time.

• **Discussion**

The ABCC-EC model is demonstrated using a proof-of-concept implementation in this research. The performance evaluation results in Figure 6.12 imply the feasibility of this approach in real-world CE-IoT platforms. As future work, the proof-of-concept implementation could be developed into a robust implementation framework that can meet the need of a rapidly growing domain, viz. the IoT, to achieve the goal of deploying the ABCC-EC model in commercial CE-IoT architectures. In this research, we utilized the enforcement architecture based on one of largest Cloud services provider, AWS and its services, viz. AWS IoT, AWS Greengrass, and AWS Lambda function. One of the relevant issues for wide-adoption of ABCC in real-world is scalability. With billions of devices and their attributes, there is an inevitable need for an ABCC policy management tool similar to the Policy Machine or XACML for ABAC, which will act as a service and provide attribute-based communication control capabilities for different applications and platforms.

CHAPTER 7: CONCLUSION AND FUTURE WORK

This chapter summaries the contributions of this dissertation and presents potential future directions of research.

7.1 Summary

This dissertation makes fundamental contributions towards developing ABAC and ABCC models for Cloud and CE-IoT architectures and promoting their application in real-world platforms. Initially, it focuses on the Cloud Computing domain where it develops a role-centric ABAC model, the UAE-OSAC model, for OpenStack, an open source Cloud IaaS platform. Similarly, focusing on the Cloud framework, it then develops a restricted HGABAC (*rHGABAC*) model. The major objectives of developing these ABAC models is to exhibit the capabilities of ABAC and stimulate their enforcement in the Cloud Computing domain utilizing existing ABAC policy specification and enforcement tool, viz., the Policy Machine (PM), together with customized implementation of the Authorization Engine (AE) developed in this research.

With the emergence of Cloud-Enabled IoT, this dissertation aims to address security and privacy issues in the CE-IoT domain. For this purpose, first, it studies AWS-IoT, a real-world instance of CE-IoT, and develops an access control model for it, called AWS-IoTAC, based on AWS's policy-based approach. However, it proposes ABAC enhancements to the AWS-IoTAC model to address dynamic and flexible access and authorization control requirements in CE-IoT. Then, it focuses on the access and communication control needs and concerns in the CE-IoT domain and relevant sub-domains (e.g., Wearable IoT (WIoT)). Inspired by a WIoT use case, it extends the ACO architecture and categorizes various interactions (accesses and data exchanges) within the enhanced ACO (EACO) into three access control models in the Access Control framework.

Finally, this dissertation recognizes the inevitable need of communication control models to enable secure data and information flow and enforce user-specific privacy policies in the CE-IoT architectures. Thus, it introduces a new concept of Attribute-Based Communication Control (ABCC)

and presents a conceptual ABCC model along with the description of its characteristics and components. In an edge computing enabled CE-IoT context, it develops an ABCC model for controlling communication and data flow between edge network of things and Cloud. The implementation of this model is presented in AWS and its IoT and edge computing service.

In summary, the technical approach adopted here is to study and develop attribute-based access and communication control models to secure the Cloud and CE-IoT arena. To motivate the application of these models in real-world scenarios, proof-of-concept implementations and feasibility analysis of the models are presented in relevant areas.

7.2 Future Work

There are several potential directions which can be studied and explored as extensions to this research. The ABAC and ABCC models developed in this dissertation could be enhanced with additional capabilities (e.g., trust mechanisms) based on user and domain requirements. Moreover, attribute-based is a simple and flexible approach and could be applied in other domains beyond access and communication control models.

This dissertation focuses on access and communication control needs within a single Cloud infrastructure. However, in the continuously broadening IoT space, multi-cloud infrastructures are becoming a necessity to support IoT. The multi-cloud IoT architecture directly implies IoT federation and trust models in CE-IoT. The concept of multi-cloud may refer to different infrastructures, such as multiple Clouds interacting with each other (e.g., AWS and OpenStack), multiple tenants sharing data and information inside a Cloud (e.g., AWS tenants), or multiple cloudlets interacting with each other within one Cloud or across multiple Clouds. The access and communication control requirements are diverse in such dynamic environment since it involves several distributed components. Therefore, further research on multi-cloud access and communication control models in the IoT arena is a viable direction.

With the introduction of Attribute-Based Communication Control (ABCC) in this research, there are various avenues to develop ABCC models further. There is a lack of knowledge and

literature on ABCC which need to be advanced with significant research on ABCC in different contexts. The scope of ABCC models needs to investigated in specific IoT domains where confidentiality and integrity of communication occurring between several components is critical. Some of the relevant IoT domains are Battlefield IoT, Medical/Healthcare IoT, and Vehicular IoT.

BIBLIOGRAPHY

[1] Amazon Web Services (AWS). https://aws.amazon.com/. Accessed: 2016-11-10.

[2] AWS Greengrass Service. https://aws.amazon.com/greengrass/. Accessed: 01/08/2018.

[3] AWS IoT Platform. http://docs.aws.amazon.com/iot/latest/developerguide/what-is-aws-iot.html. Accessed: 2017-01-08.

[4] AWS Lambda Service. https://aws.amazon.com/lambda/. Accessed: 01/08/2017.

[5] AWS Python SDK. https://github.com/aws/aws-iot-device-sdk-python. Accessed: 01/08/2017.

[6] AWS SDK for JavaScript in Node.js. https://aws.amazon.com/sdk-for-node-js/. Accessed: 2016-08-10.

[7] Azure IoT. https://azure.microsoft.com/en-us/overview/iot/. Accessed: 2017-01-15.

[8] Build Your Blueprint for the Internet of Things, Based on Five Architecture Styles. https://www.gartner.com/doc/2854218/build-blueprint-internet-things-based. Accessed: 2017-01-02.

[9] Fitness Trackers Could Benefit from Better Security. http://www.ed.ac.uk/news/2017/fitness-trackers-could-benefit-from-better-securit. Accessed: 2017-09-15.

[10] Gartner Says 8.4 Billion Connected "Things" Will Be in Use in 2017, Up 31 Percent From 2016. https://www.gartner.com/newsroom/id/3598917. Accessed: 2017-03-15.

[11] Google Cloud Platform. https://cloud.google.com/. Accessed: 2016-12-10.

[12] Google Internet of Things. https://cloud.google.com/solutions/iot-overview/. Accessed: 2016-12-10.

[13] Guard - Information Security. https://en.wikipedia.org/wiki/Guard_(information_security). Accessed: 2018-06-02.

[14] Harmonia-1.5. https://github.com/PM-Master/Harmonia-1.5. Accessed: 2016-08-18.

[15] Harmonia-1.6. https://github.com/PM-Master/Harmonia-1.6. Accessed: 2017-01-18.

[16] Here's How the Internet of Things (IoT) Will Change Workplaces. http://www.insight.com/en_US/learn/content/2017/02072017-heres-how-the-internet-of-things-iot-will-change-workplaces.html. Accessed: 2017-06-10.

[17] How The Internet of Things is Revolutionizing Manufacturing. http://www.businessinsider.com/internet-of-things-in-manufacturing-2016-10. Accessed: 2017-08-04.

[18] IoT - Internet of Things. http://www.webopedia.com/TERM/I/internet_of_things.html. Accessed: 2017-01-04.

[19] IoT Device Categories Class 0,1,2. http://www.cisoplatform.com/profiles/blogs/classification-of-iot-devices. Accessed: 2017-08-10.

[20] Living in the Cloud, Gateway or on the Edge: IoT's Fragmented Future. http://www.wi-next.com/2015/03/living-cloud-gateway-edge-iots-fragmented-future/. Accessed: 2017-01-02.

[21] Microsoft Azure. https://azure.microsoft.com/en-us/. Accessed: 2016-11-28.

[22] MQTT.fx - A JavaFX based MQTT Client. http://www.mqttfx.org/. Accessed: 2016-09-10.

[23] Nymi Band. https://nymi.com/. Accessed: 2017-01-08.

[24] OpenStack. https://www.openstack.org/. Accessed: 2016-11-10.

[25] Policy Machine. http://csrc.nist.gov/pm/. Accessed: 2016-09-10.

[26] The Challenges of Wearable Electronics. http://www1.futureelectronics.com/Mailing/etechs/TEConnectivity/etechALERT_TE_ConsumerWearables/Images/WearablesWhitePaper_TE.pdf. Accessed: 2017-08-08.

[27] Top 6 Wearable Safety Devices. https://blog.cammy.com/top-wearable-safety-devices. Accessed: 2017-01-08.

[28] Transport Layer Security. https://tools.ietf.org/html/rfc5246. Accessed: 2017-08-10.

[29] Wearable Technologies. https://www.wearable-technologies.com/innovation-worldcup/categories/. Accessed: 2017-08-15.

[30] What is AWS Greengrass? https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html. Accessed: 01/08/2018.

[31] X.509 Certificates. http://searchsecurity.techtarget.com/definition/X509-certificate. Accessed: 2017-02-10.

[32] XACML. https://en.wikipedia.org/wiki/XACML. Accessed: 2016-10-10.

[33] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[34] Mohammad A Al-Kahtani and Ravi Sandhu. A Model for Attribute-Based User-Role Assignment. In *Proceedings of 18th Annual Computer Security Applications Conference*, pages 353–362. IEEE, 2002.

[35] Asma Alshehri and Ravi Sandhu. Access Control Models for Cloud-Enabled Internet of Things: A Proposed Architecture and Research Agenda. In *International Conference on Collaboration and Internet Computing (CIC)*, pages 530–538, 2016.

[36] Asma Alshehri and Ravi Sandhu. Access Control Models for Virtual Object Communication in Cloud-Enabled IoT. In *International Conference on Information Reuse and Integration (IRI)*, pages 16–25. IEEE, 2017.

[37] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010.

[38] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. An Atribute-Based Access Control Extension for OpenStack and its Enforcement Utilizing the Policy Machine. In *2nd International Conference on Collaboration and Internet Computing (CIC)*, pages 37–45. IEEE, 2016.

[39] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 17–28. ACM, 2017.

[40] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. Access Control Model for AWS Internet of Things. In *International Conference on Network and System Security*, pages 721–736. Springer, 2017.

[41] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. An Access Control Framework for Cloud-Enabled Wearable Internet of Things. In *3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 328–338. IEEE, 2017.

[42] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. A Comparison of Logical-formula and Enumerated Authorization Policy ABAC Models. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 122–129. Springer, 2016.

[43] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. Label-Based Access Control: An ABAC Model with Enumerated Authorization Policy. In *Proceedings of the ACM International Workshop on Attribute-Based Access Control*, pages 1–12. ACM, 2016.

[44] Alessio Botta, Walter De Donato, Valerio Persico, and Antonio Pescapé. On the Integration of Cloud Computing and Internet of Things. In *International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 23–30. IEEE, 2014.

[45] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of Cloud Computing and Internet of Things: A Survey. *Future Generation Computer Systems*, 56:684–700, 2016.

[46] David W Chadwick, Kristy Siu, Craig Lee, Yann Fouillat, and Damien Germonville. Adding Federated Identity Management to OpenStack. *Journal of Grid Computing*, 12(1):3–27, 2014.

[47] Luis Cruz-Piris, Diego Rivera, German Lopez-Civera, Enrique De la Hoz, Ivan Marsa-Maestre, and Juan R Velasco. Protecting Sensors in an IoT Environment by Modelling Communications as Resources. *Multidisciplinary Digital Publishing Institute Proceedings*, 1(8):801, 2017.

[48] Luis Cruz-Piris, Diego Rivera, Ivan Marsa-Maestre, Enrique de la Hoz, and Juan R Velasco. Access Control Mechanism for IoT Environments Based on Modelling Communication Procedures as Resources. *Sensors*, 18(3):917, 2018.

[49] Li Da Xu, Wu He, and Shancang Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[50] Angelika Dohr, Robert Modre-Opsrian, Mario Drobics, Dieter Hayn, and Günter Schreier. The Internet of Things for Ambient Assisted Living. In *Seventh International Conference on Information Technology: New Generations (ITNG)*, pages 804–809. IEEE, 2010.

[51] David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The Policy Machine: A Novel Architecture and Framework for Access Control Policy Specification and Enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

[52] David Ferraiolo, Serban Gavrila, and Wayne Jansen. Policy Machine: Features, Architecture, and Specification. *National Institute of Standards and Technology Internal Report 7987*, 2014.

[53] David F Ferraiolo, Larry Feldman, and Gregory A Witte. Exploring the Next Generation of Access Control Methodologies. Technical report, 2016.

[54] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.

[55] Ludwig Fuchs, Günther Pernul, and Ravi Sandhu. Roles in Information Security–A Survey and Classification of the Research Area. *Computers & Security*, 30(8):748–769, 2011.

[56] 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. ACM, 2006.

[57] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[58] Maanak Gupta and Ravi Sandhu. The GURA_G Administrative Model for User and Group Attribute Assignment. In *International Conference on Network and System Security*, pages 318–332. Springer, 2016.

[59] Sergio Gusmeroli, Salvatore Piccione, and Domenico Rotondi. A Capability-Based Security Approach to Manage Access Control in the Internet of Things. *Mathematical and Computer Modelling*, 58(5):1189–1205, 2013.

[60] Michael Hanspach and Jorg Keller. In Guards We Trust: Security and Privacy in Operating Systems Revisited. In *International Conference on Social Computing (SocialCom), 2013*, pages 578–585. IEEE, 2013.

[61] José L Hernández-Ramos, Antonio J Jara, Leandro Marin, and Antonio F Skarmeta. Distributed Capability-Based Access Control for the Internet of Things. *Journal of Internet Services and Information Security (JISIS)*, 3(3/4):1–16, 2013.

[62] Shivayogi Hiremath, Geng Yang, and Kunal Mankodiya. Wearable Internet of Things: Concept, Architectural Components and Promises for Person-Centered Healthcare. In *EAI 4th International Conference on Wireless Mobile Communication and Healthcare (Mobihealth)*, pages 304–307. IEEE, 2014.

[63] Vincent C Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to Attribute Based Access Control (ABAC) Definition and Considerations. *NIST Special Publication 800-162*, 2014.

[64] Vincent C Hu, D Richard Kuhn, and David F Ferraiolo. Attribute-Based Access Control. *IEEE Computer*, 48(2):85–88, 2015.

[65] Junbeom Hur and Dong Kun Noh. Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2011.

[66] SM Riazul Islam, Daehan Kwak, MD Humaun Kabir, Mahmud Hossain, and Kyung-Sup Kwak. The Internet of Things for Health Care: A Comprehensive Survey. *IEEE Access*, 3:678–708, 2015.

[67] Xin Jin, Ram Krishnan, and Ravi Sandhu. A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer, 2012.

[68] Xin Jin, Ram Krishnan, and Ravi Sandhu. Role and Attribute Based Collaborative Administration of Intra-Tenant Cloud IaaS. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 261–274. IEEE, 2014.

[69] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: Role-Centric Attribute-Based Access Control. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 84–96. Springer, 2012.

[70] VJ Jincy and Sudharsan Sundararajan. Classification Mechanism for IoT Devices Towards Creating A Security Framework. In *Intelligent Distributed Computing*, pages 265–277. Springer, 2015.

[71] Sun Kaiwen and Yin Lihua. Attribute-Role-Based Hybrid Access Control in the Internet of Things. In *Asia-Pacific Web Conference*, pages 333–343. Springer, 2014.

[72] Rafiullah Khan, Sarmad Ullah Khan, Rifaqat Zaheer, and Shahid Khan. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. In *10th International Conference on Frontiers of Information Technology (FIT)*, pages 257–260. IEEE, 2012.

[73] Konstantinos Kotis and Artem Katasonov. An IoT-Ontology for The Representation of Interconnected, Clustered and Aligned Smart Entities. *Technical report, VTT Technical Research Center, Finland*, 2012.

[74] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding Attributes to Role-Based Access Control. *IEEE Computer*, 43(6):79–81, 2010.

[75] Craig A Lee and Nehal Desai. Approaches for Virtual Organization Support in OpenStack. In *International Conference on Cloud Engineering (IC2E)*, pages 432–438. IEEE, 2014.

[76] Jin Li, Qian Wang, Cong Wang, and Kui Ren. Enhancing Attribute-Based Encryption with Attribute Hierarchy. *Mobile Networks and Applications*, 16(5):553–561, 2011.

[77] Jing Liu, Yang Xiao, and CL Philip Chen. Authentication and Access Control in the Internet of Things. In *32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 588–592. IEEE, 2012.

[78] Parikshit N Mahalle, Bayu Anggorojati, Neeli Rashmi Prasad, and Ramjee Prasad. Identity Establishment and Capability Based Access Control (IECAC) Scheme for Internet of Things. In *15th Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 187–191. IEEE, 2012.

[79] Wu Miao, T LU, F LING, et al. Research on the Architecture of Internet of Things [C]. In *Proceedings of the 3rd International Conference on Advanced Computer Theory and Engineering: August*, pages 20–22. sn, 2010.

[80] Jiwan Ninglekhu and Ram Krishnan. AARBAC: Attribute-Based Administration of Role-based Access Control. In *3rd International Conference on Collaboration and Internet Computing (CIC)*, pages 126–135. IEEE, 2017.

[81] Jiwan Ninglekhu and Ram Krishnan. Attribute Based Administration of Role Based Access Control: A Detail Description. *arXiv preprint arXiv:1706.03171*, 2017.

[82] 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, 2016.

[83] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-Based Encryption with Non-Monotonic Access Structures. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 195–203. ACM, 2007.

[84] Aafaf Ouaddah, Hajar Mousannif, Anas Abou Elkalam, and Abdellah Ait Ouahman. Access Control in The Internet of Things: Big Challenges and New Opportunities. *Computer Networks*, 112:237–262, 2017.

[85] Pritee Parwekar. From Internet of Things Towards Cloud of Things. In *2nd International Conference on Computer and Communication Technology (ICCCT)*, pages 329–333. IEEE, 2011.

[86] Pawani Porambage, Mika Ylianttila, Corinna Schmitt, Pardeep Kumar, Andrei Gurtov, and Athanasios V Vasilakos. The Quest for Privacy in the Internet of Things. *IEEE Cloud Computing*, 3(2):36–45, 2016.

[87] Torsten Priebe, Wolfgang Dobmeier, and Nora Kamprath. Supporting Attribute-Based Access Control with Ontologies. In *First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8–pp. IEEE, 2006.

[88] Navid Pustchi and Ravi Sandhu. MT-ABAC: A Multi-Tenant Attribute-Based Access Control Model with Tenant Trust. In *International Conference on Network and System Security*, pages 206–220. Springer, 2015.

[89] 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.

[90] BB Prahlada Rao, Paval Saluia, Neetu Sharma, Ankit Mittal, and Shivay Veer Sharma. Cloud Computing for Internet of Things & Sensing Based Applications. In *Sixth International Conference on Sensing Technology (ICST)*, pages 374–380. IEEE, 2012.

[91] Rodrigo Roman, Jianying Zhou, and Javier Lopez. On the Features and Challenges of Security and Privacy in Distributed Internet of Things. *Computer Networks*, 57(10):2266–2279, 2013.

[92] Ravi Sandhu. Role-Based Access Control. *Advances in Computers*, 46:237–286, 1998.

[93] Ravi Sandhu, Edward J Coyne, Hal Feinstein, and Charles Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[94] Ravi S Sandhu and Pierangela Samarati. Access Control: Principle and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[95] Mahadev Satyanarayanan. The Emergence of Edge Computing. *Computer*, 50(1):30–39, 2017.

[96] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The Case for VM-based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*, 8(4):14–23, 2009.

[97] Mahadev Satyanarayanan, Zhuo Chen, Kiryong Ha, Wenlu Hu, Wolfgang Richter, and Padmanabhan Pillai. Cloudlets: At the Leading Edge of Mobile-Cloud Convergence. In *6th International Conference on Mobile Computing, Applications and Services (MobiCASE)*, pages 1–9. IEEE, 2014.

[98] Mahadev Satyanarayanan, Pieter Simoens, Yu Xiao, Padmanabhan Pillai, Zhuo Chen, Kiryong Ha, Wenlu Hu, and Brandon Amos. Edge Analytics in the Internet of Things. *IEEE Pervasive Computing*, 14(2):24–31, 2015.

[99] Hans Schaffers, Nicos Komninos, Marc Pallot, Brigitte Trousse, Michael Nilsson, and Alvaro Oliveira. Smart Cities and The Future Internet: Towards Cooperation Frameworks for Open Innovation. *The Future Internet*, pages 431–446, 2011.

[100] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. OpenStack: Toward an Open-Source Solution for Cloud Computing. *International Journal of Computer Applications*, 55(3), 2012.

[101] Daniel Servos and Sylvia L Osborn. HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control. In *International Symposium on Foundations and Practice of Security*, pages 187–204. Springer, 2014.

[102] Pallavi Sethi and Smruti R Sarangi. Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering*, 2017.

[103] Hai-bo Shen and Fan Hong. An Attribute-Based Access Control Model for Web Services. In *Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'06)*, pages 74–79. IEEE, 2006.

[104] George Suciu, Alexandru Vulpe, Simona Halunga, Octavian Fratu, Gyorgy Todoran, and Victor Suciu. Smart Cities Built on Resilient Cloud Computing and Secure Internet of Things. In *19th International Conference on Control Systems and Computer Science (CSCS)*, pages 513–518. IEEE, 2013.

[105] Bo Tang and Ravi Sandhu. Extending OpenStack Access Control with Domain Trust. In *International Conference on Network and System Security*, pages 54–69. Springer, 2014.

[106] Panagiotis Vlacheas, Raffaele Giaffreda, Vera Stavroulaki, Dimitris Kelaidonis, Vassilis Foteinos, George Poulios, Panagiotis Demestichas, Andrey Somov, Abdur Rahim Biswas, and Klaus Moessner. Enabling Smart Cities Through a Cognitive Management Framework for The Internet of Things. *IEEE Communications Magazine*, 51(6):102–111, 2013.

[107] Jeffrey Voas. Networks of 'Things'. *NIST Special Publication*, 800(183):800–183, 2016.

[108] Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A Logic-Based Framework for Attribute Based Access Control. In *Proceedings of the ACM Workshop on Formal Methods in Security Engineering*, pages 45–55. ACM, 2004.

[109] Evan Welbourne, Leilani Battle, Garrett Cole, Kayla Gould, Kyle Rector, Samuel Raymer, Magdalena Balazinska, and Gaetano Borriello. Building The Internet of Things Using RFID: The RFID Ecosystem Experience. *IEEE Internet Computing*, 13(3), 2009.

[110] Miao Wu, Ting-Jie Lu, Fei-Yang Ling, Jing Sun, and Hui-Ying Du. Research on The Architecture of Internet of Things. In *3rd International Conference on Advanced Computer Theory and Engineering (ICACTE)*, volume 5, pages V5–484. IEEE, 2010.

[111] Zhihong Yang, Yingzhao Yue, Yu Yang, Yufeng Peng, Xiaobo Wang, and Wenji Liu. Study and Application on The Architecture and Key Technologies for IoT. In *International Conference on Multimedia Technology (ICMT)*, pages 747–751. IEEE, 2011.

[112] Eric Yuan and Jin Tong. Attributed Based Access Control (ABAC) for Web Services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[113] Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of Things for Smart Cities. *IEEE Internet of Things Journal*, 1(1):22–32, 2014.

[114] Guoping Zhang and Jiazheng Tian. An Extended Role Based Access Control Model for the Internet of Things. In *International Conference on Information Networking and Automation (ICINA)*, volume 1, pages V1–319. IEEE, 2010.

[115] Yun Zhang, Farhan Patwa, and Ravi Sandhu. Community-Based Secure Information and Resource Sharing in AWS Public Cloud. In *International Conference on Collaboration and Internet Computing (CIC)*, pages 46–53. IEEE, 2015.

[116] Yun Zhang, Farhan Patwa, Ravi Sandhu, and Bo Tang. Hierarchical Secure Information and Resource Sharing in OpenStack Community Cloud. In *International Conference on Information Reuse and Integration (IRI)*, pages 419–426. IEEE, 2015.

VITA

Smriti Bhatt grew up in Mahendranagar, Nepal. She earned a Bachelor of Engineering (B.E.) in Computer Engineering from Kathmandu University, Dhulikhel, Nepal in 2011. After completing her engineering, she worked for two years as a Software Engineer in Nepal at Verisk Information Technologies, a subsidiary of Verisk Analytics. In Fall 2013, Smriti started pursuing her doctoral degree at University of Texas at San Antonio. She joined the Institute for Cyber Security (ICS), Department of Computer Science, UTSA in 2015 and started doing research under the supervision of Dr. Ravi Sandhu. She has received her Masters in Computer Science (M.S.) with a concentration in Computer and Information Security from UTSA. Her research area is Cloud and Internet of Things Security mainly focused on Attribute-Based Access and Communication Control models and their application in different domains within the context of Cloud Computing and Cloud-Enabled IoT.