SECURE CLOUD ASSISTED SMART CARS AND BIG DATA: ACCESS CONTROL MODELS AND IMPLEMENTATION

by

MAANAK GUPTA, 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 B. White, Ph.D.
Ram Krishnan, Ph.D.

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

DEDICATION

This dissertation is dedicated to my parents Dr. Surinder Gupta and Mrs. Meenu Gupta, who offered unconditional support and are a big source of inspiration. I will always be indebted for the love and compassion they have bestowed on me. This work is also dedicated to my wife Prachi, son Maanvik and brother Ashutosh Gupta, who motivated and encouraged me through out the most challenging phase of my life. Undoubtedly, without Prachi, our journey this far would have not been possible. The blessings of almighty, my grandparents, family and friends is the real force because of which I am able to write this piece!

ACKNOWLEDGEMENTS

I express my deepest gratitude and respect to my advisor and mentor Professor Ravi Sandhu, who supervised me through this demanding journey. I have no hesitation in saying that without his able guidance, thoughtful comments and incessant encouragement I would not have come this far. He was the torchbearer who pushed me to critical thinking and exploring creative solutions for real world problems. The knowledge and experience I gained professionally and personally from him will be remembered, and will set course for my career and life. He always acknowledged my untimely requests, corrected me on occasions and supported me in decisions made during research and beyond. I will always be indebted for his modesty, humility and wisdom. It is an honour to work under Dr. Sandhu and I will always relish these days.

A big thank you to Mr. Farhan Patwa, Associate Director and Chief Architect at the Institute for Cyber Security (ICS), UTSA, for his insightful comments and incessant support during my doctoral studies. His humble temperament and sound technical knowledge are uncommon to find. I am very fortunate to have known and worked with him. Mr. James Benson is another outstanding person, I have met at ICS. He helped me in AWS implementation of our work in smart connected cars. His relentless technical support was absolutely needed to sail through this journey. I am privileged to work with the best research team and the staff at ICS which together brought me in a position to write my Ph.D. dissertation.

I would like to thank my doctoral dissertation committee members Dr. Murtuza Jadliwala, Dr. Palden Lama, Dr. Gregory B. White and Dr. Ram Krishnan for their thoughtful comments, suggestions and time. Their observations have made this research work more valuable and interesting.

I thank the staff members at ICS, Suzanne Tanaka and Lisa Ho for all the administrative work they do for us. You both are most efficient, amazing and great. The staff in computer science department especially Susan Allen, were a big help during my doctoral studies.

I am thankful to my mother-in-law Mrs. Sunita Bansal, who supported and encouraged me. She always stood with me during difficult and tough times.

I sincerely appreciate my peers and friends at UTSA, my lab fellows and seniors who assisted

me throughout my doctoral work. You all were a big support and constant help during this work. I acknowledge all federal and private agencies including National Science Foundation (NSF), Department of Defense (DoD) and other organizations for providing grants for my research.

To anyone I may have missed, you are by no means any less special and I am grateful for all your help and support.

Once again, to all the wonderful people I have in my life, thank you!

This Masters Thesis/Recital Document or Doctoral Dissertation was produced in accordance with guidelines which permit the inclusion as part of the Masters Thesis/Recital Document or Doctoral Dissertation the text of an original paper, or papers, submitted for publication. The Masters Thesis/Recital Document or Doctoral Dissertation must still conform to all other requirements explained in the Guide for the Preparation of a Masters Thesis/Recital Document or Doctoral Dissertation at The University of Texas at San Antonio. It must include a comprehensive abstract, a full introduction and literature review, and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

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

December 2018

SECURE CLOUD ASSISTED SMART CARS AND BIG DATA:
ACCESS CONTROL MODELS AND IMPLEMENTATION

Maanak Gupta, Ph.D.
The University of Texas at San Antonio, 2018

Supervising Professor: Ravi Sandhu, Ph.D.

Access control security mechanisms, including discretionary-DAC, mandatory-MAC and role based-RBAC, help to restrict unauthorized access and operations on data and other resources in computer systems. More recently, attribute based access control (ABAC) has been proposed to provide flexibility and fine grained authorization based on the attributes of users, resources, and other relevant entities. Hierarchical group and attribute based access control (HGABAC) model has been proposed to offer administrative benefits in ABAC system by introducing groups, which enable multiple attributes assignment and removal from its member users or objects with single administrative action. However, the administration of HGABAC, including who will assign users to groups, or what attributes are inherited or directly assigned, and what attributes an entity will get based on set of administrative rules, are not addressed so far.

Besides developing the foundational aspects of ABAC, it is also important to understand its applicability in real problems which can impact our society. Smart cars are among the essential components and major drivers of future cities and connected world. The interaction among connected entities in this vehicular internet of things (IoT) domain, which also involves smart traffic infrastructure, restaurant beacons, emergency vehicles, etc., will offer many real-time service applications and provide safer and more pleasant driving experience to consumers. With more than 100 million lines of code and hundreds of sensors on board generating huge amounts of data, these vehicles are often termed as 'datacenter on wheels'. These connected vehicles (CVs) expose a large attack surface, which can be remotely compromised and exploited by malicious attackers. Security and privacy are big concerns that deter the adoption of smart cars, which if not properly addressed will have grave implications with risk to human life and limb. Also, the recent

data breaches and growing privacy concerns of consumer data further pushes the need for stronger security mechanisms for Big Data.

In this dissertation, we investigate and develop both the foundational and application aspects of ABAC models. First, we present an administrative model for HGABAC, referred as GURA_G, which defines three sub models user attribute assignment (UAA), user group attribute assignment (UGAA) and user to group assignment (UGA), for adding and removing attributes from users and groups along with user groups membership. As it is important to understand what attributes a user will get based on a set of administrative rules, we present reachability analysis for restricted form of GURA_G model, called rGURA_G. In general the problem is PSPACE-complete, however for certain cases polynomial time algorithms have been devised.

Second, we investigate the smart cars ecosystem and propose an authorization framework to secure this dynamic and distributed system where interaction among vehicle and infrastructures is not pre-defined. We provide an extended access control oriented (E-ACO) architecture relevant to connected vehicles and discuss the need of vehicular clouds in this time and location sensitive environment. We also develop dynamic groups and attribute-based access control (ABAC) model (referred as CV-ABAC_G) to secure communication, data exchange and resource access in smart vehicles ecosystem. This model takes into account the user-centric privacy preferences along with system-defined policies to make access decisions. We propose a novel concept of groups in context of cloud assisted smart cars, which are dynamically assigned to moving entities like vehicles, based on their current GPS coordinates, direction or other attributes, to ensure relevance of location and time sensitive notification services offered to drivers, and to provide administrative benefits to manage large numbers of entities and enable attributes inheritance for fine grained authorization.

Finally, as all IoT devices and smart cars produce enormous amounts of data which is sent to central cloud for processing and storage, it is imperative to understand and develop authorization solutions for most widely used Big data processing platform, Hadoop. Henceforth, we first formalize the current access control model for Hadoop ecosystem, called HeAC. We then extend this model to provide a cohesive object-tagged role-based access control (OT-RBAC) model, consis-

tent with generally accepted academic concepts of RBAC. We also present a fine-grained attribute-based access control model, referred as HeABAC, catering to the security and privacy needs of multi-tenant Hadoop ecosystem.

In closing, we conclude this dissertation and provide some future work directions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Internet of Things (IoT) has become a dominant technology which has proliferated to different application domains including health-care, homes, industry, and power-grid, to make lives smarter. It is predicted [22] that the global IoT market will grow to $457 Billion by year 2020, attaining a compound annual growth rate of 28.5%. Automation is leading the world today, and with 'things' around sensing and acting on their own or with a remote user command, humans have anything accessible with a finger touch. Data generated by these smart devices unleash countless business opportunities and offer customer targeted services. IoT along with 'infinite' capabilities of cloud computing are ideally matched with desirable synergy in current technology-oriented world, termed as cloud-enabled, cloud-centric or cloud-assisted IoT in literature [34, 48, 49, 118].

IoT is embraced by every industry with automobile manufacturers and transportation among the most aggressive. The vision of smart city incorporates intelligent transportation where connected vehicles can 'talk' to each other (V2V) and exchange information to ensure driver safety and offer location-based services. These intelligent vehicles can also interact with smart roadside infrastructure (V2I), with pedestrian on road (V2H) or send data to the cloud for processing. A car will receive information about nearby parking garages, restaurant offers or remote engine monitoring by authorized mechanic with nearby repair facility and discounts updating automatically. These services will provide pleasant travel experience to drivers and unleash business potential in this intelligent transportation domain. Smart internet connected vehicles embed software having more than 100 million lines of code to control critical systems and functionality, with plethora of sensors and electronic control units (ECUs) on board generating huge amounts of data, so these vehicles are often termed as 'datacenter on wheels'.

Security and privacy have been a serious concern and challenge for the adoption of IoT. The gravity of these issues is magnified when we think about implications in smart cars and the emerging concept of autonomous vehicles. With over 100 millions lines of code, more than 100 electronic control units (ECUs) and broad attack surface opened by features such as onboard diag-

nostics, driver assistance systems and airbags, it becomes a challenge to protect this smart entity. Further, the data sent to the cloud for processing and storage also have security challenges which are escalating with the continuous surge in data breaches around the globe.

This dissertation focuses on the development and implementation of access control models for securing unauthorized access to cloud assisted Smart Cars and Big Data, and evolving a family of models from conventional role based to fine grained attribute based access control. The work also presents foundational facets of group supported ABAC and provides administrative models, referred as GURA_G along with attribute reachability analysis.

1.1 Motivation

Access control [72, 144, 145] mechanisms are widely used to restrict unauthorized access to resources and secure data exchange among entities. Series of access control models have evolved over time starting with Lampson's access matrix in late 1960's to discretionary access control (DAC) [145], mandatory access control (MAC) [143], role based access control (RBAC) [72, 144] and most recently to attribute based access control (ABAC) [97, 99, 105]. The growing need for ABAC is driven by the change in current technologies and computer systems from conventional enterprise style applications to more pervasive and distributed domains including IoT, Smart Cars, Blockchain, Software Defined Networks, Cloud Computing etc. which cannot be secured by single administrative domain, and have characteristics (like location, IP address or time) of the entities that play an important role to determine the fine grained privileges. However, the concerns of administration of attributes of different entities still is an important question, which can be termed as attribute engineering in ABAC as a counterpart of role engineering in RBAC.

Intelligent transportation and future smart cities will change the world. These revolutionary visions have smart cars as essential component which will enable notifications, alerts and advertisements to the drivers and collection of the generated sensor data in remote cloud infrastructures. Further, the connectivity of the car to the internet opens endless opportunities including remote diagnostics, over the air (OTA) updates for the firmware, or e-tolls. Vehicles can receive speed limit

notification and flash flood alerts on car dashboard or via seat vibration. A car will receive information about nearby parking garages, restaurant offers or remote engine monitoring by authorized mechanic with nearby repair facility and discounts updating automatically.

As vehicles get exposed to external environment and internet, they become vulnerable to cyber attacks. Common security vulnerabilities including buffer overflow, malware, privilege escalation, and trojans etc. can be exploited in connected vehicles. Other potential threats include untrustworthy or fake messages from smart objects, malicious software injection, data privacy, ECU hacking and control, and spoofing connected vehicle sensor. With broad attack surface exposed via air-bag ECU, On-Board Diagnostics (OBD) port, USB, Bluetooth, remote key, and tire-pressure monitoring system etc. these attacks have become much easier to orchestrate. In-vehicle Controller Area Network (CAN) bus also needs security to protect message exchange among ECUs. Further, communication with external networks including cellular, WiFi and insecure public networks of gas stations, toll roads, service garages, or after-market dongles are a big threat to connected vehicles security. Cyber incidents including Jeep [182] and Tesla Model X [172] hacks where engine was stopped and steering remotely controlled demonstrate security vulnerabilities. Such incidents have serious implications as they can result in loss of life. Securing Internet of Vehicles (IoV) and smart cars will require protecting control systems (on-board diagnostic (OBD) port, CAN bus etc.), protecting infotainment systems, securing smartphone applications, securing infrastructure, securing over-the air updates, and securing hardware from manual tampering. Security become hard to implement considering IoV characteristics like dynamic topology, mobility, and distributed network.

Smart cars ecosystem involves dynamic interaction and message exchange among connected objects, which must be authorized. It is necessary that only legitimate entities are allowed to control on-board sensors, data messages and receive notifications. Smart cars location-based services enable notifications and alerts to vehicles. A user must be allowed to set his personal preferences whether he wants to receive advertisements or filter out which ones are acceptable or who can access their car's sensors. For instance, a user may not want to receive restaurant notifications but is interested in flash-flood warnings. System wide policy, like a speed warning to all over-speeding

vehicles or a policy of who can control speed of autonomous car are needed. Vehicles exchange basic safety messages (BSMs) which raises an important question about trust. It must be ensured that information received is correct and from a trusted party, before being used by on-vehicle applications. Applications access sensors within and outside the car, which must be authorized, for example, a lane departure warning system accessing tire sensors must be checked to prevent a spoofed application reading vehicle movements. A passenger accessing infotainment (information and entertainment) systems of car via Bluetooth or using smartphone inside car must also be authorized. Further, data protection in cloud is critical due to frequent occurrence of data breaches. Big Data access control is essential when user privacy has to be ensured and unauthorized disclosure is not allowed. As cars produce data which is stored in cloud servers, it becomes imperative to secure data at rest, where Hadoop is the predominant big data framework for storage and processing.

To solve these issues, it is important to have a ladder approach and understand problems before offering comprehensive solutions. To start, an authorization framework is required to recognize the various architecture layers to illustrate different interaction and data exchange scenarios in vehicular IoT and to propose access control models at various layers including physical, virtual objects, cloud layer and applications. Attribute-based access control (ABAC) provides finer granularity and offers flexibility in distributed multi-entity communication scenarios, which is needed to address access control concerns in cloud assisted smart cars and Hadoop big data processing framework.

1.2 Problem Statement

New and emerging technologies such as smart connected cars and multi-tenant big data platforms require innovative access control models.

1.3 Thesis Statement

The established paradigms of role-based and attribute-based access control can be adapted and extended to provide fine-grained and dynamic authorization for cloud assisted smart cars and Hadoop big data framework.

1.4 Scope and Assumptions

The scope of this dissertation is to first develop foundational aspects of access control particularly related to groups based ABAC and their application in technology domains including smart cars and big data. Some of the assumptions taken during this work, and its limitations, are as follows.

- The GURA_{G} model assumes that the group hierarchy in the system is predefined by the security architect and the administrative model doesn't change the hierarchy. Further, the model is only defined for user attributes and a similar model can be defined for objects also.

- The polynomial algorithms provided for decidable reachability analysis are not claimed to be the only and most optimal solutions. It is expected that more optimized algorithms can be developed and is part of the future work.

- Multi-cloud collaboration between edge cloudlets and/or central clouds is not considered with the assumption of single cloud for a defined geographical area in smart vehicles.

- Connected vehicle $\text{CV-ABAC}_{\text{G}}$ model assumes that the information and attributes shared by source and object entities are trusted, for instance, location coordinates sent by a car are correct, and uses this shared information to make access decision. How to ensure that the information is from a trusted source or correct is out of scope.

- Direct physical access and tampering of on-board units and sensors in connected smart vehicles is not considered, although they also need security from unwanted access.

- In-vehicle security is not considered in this dissertation which provides a two layer security in case the external interface is compromised. Further, data in flow security from smart cars to central cloud or big data frameworks is beyond the scope.

- Several big data frameworks are available but this dissertation primary focusses on Hadoop ecosystem which is an important and widely used framework by academia and industry.

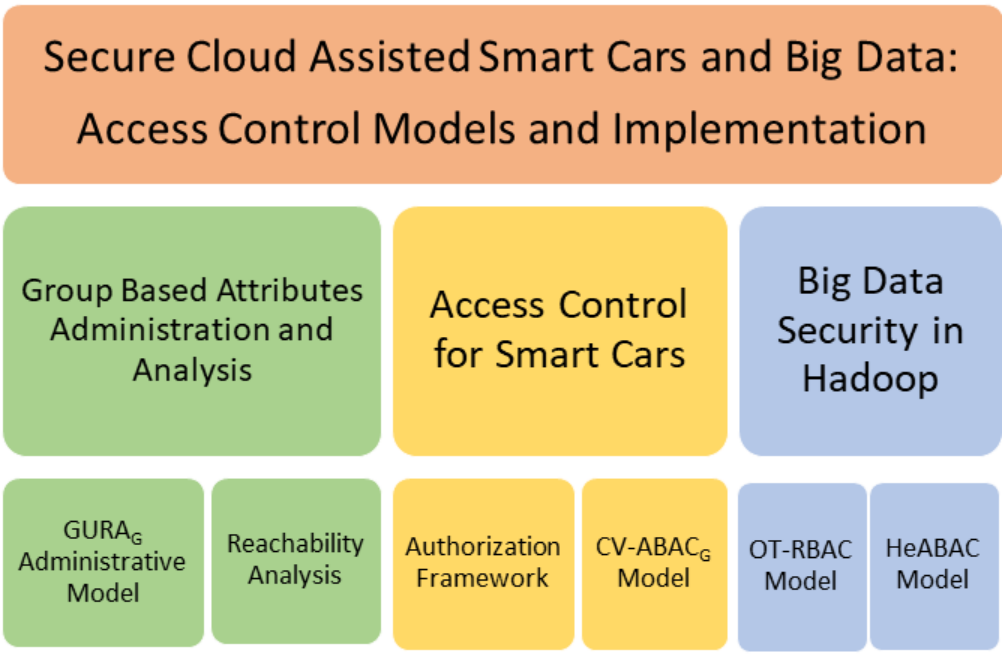


Figure 1.1: Dissertation Contributions

- Data ingestion into Hadoop cluster is beyond the scope of this work and for the access control points discussed, we assume data is already present inside the cluster. Also, we ignore deny access request, for the sake of simplicity.

1.5 Summary of Contributions

The contributions of this dissertation are shown in Figure 1.1 as discussed below:

- **User attributes administration in group supported ABAC and reachability analysis:** An administrative model GURA$_G$ for user attribute assignment based on the direct and inherited values from member groups is presented. The model has three sub-models for user attribute assignment (UAA), user to group assignment (UGA) and user group attribute assignment (UGAA). Further, as the attributes of user determines its permissions, this work also presents reachability analysis of to determine the attributes a user can achieve with a predetermined set of rules defined in GURA$_G$ model.

- **Access control solutions for smart cars:** Next, the dissertation proposes an extended access control oriented architecture (E-ACO) for connected cars and discusses an authoriza-

tion framework to provide access control requirements at various layers defined in E-ACO. Further, it also presents a dynamic groups and location based ABAC model referred as CV-ABAC_G to secure access and communication in dynamic and location centric smart cars ecosystem. The model takes into consideration user privacy preferences along with other system wide policies which are used to determine access and communication decision. The groups are dynamically assigned to moving vehicles based on their GPS locations, speed or their attributes to ensure relevance of notifications and services to smart vehicles and also restrict access and communication with resources onboard.

- **Access control for multi-tenant Hadoop ecosystem:** Lastly, the dissertation proposes a family of access control models for widely used big data processing and storage framework called Hadoop ecosystem. Since smart cars produce enormous amounts of data which is stored in cloud supported big data frameworks, the dissertation presents object tagged RBAC (OT-RBAC) and ABAC models (referred as HeABAC) for Hadoop ecosystem to protect access to resources and data in this multi-tenant distributed system.

1.6 Organization of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 discusses relevant background in context of access control models, smart cars security and Hadoop. Chapter 3 proposes formal GURA_G model for groups based user attributes administration and discusses three sub-models for user attribute assignment (UAA), user group attribute assignment (UGAA) and user to user group assignment (UGA). The chapter also discusses the reachability analysis for user attributes for a predefined set of administrative rules stated in restricted GURA_G model. Chapter 4 presents an authorization framework for connected vehicles together with an extended access control oriented architecture. Dynamic groups and attribute based access control model, called as CV-ABAC_G, along with real world use-cases and performance analysis are also discussed in this chapter. Chapter 5 presents Object-Tagged RBAC (OT-RBAC) model and attribute based model (HeABAC) for Hadoop ecosystem. Chapter 6 summarizes the dissertation with future work directions.

CHAPTER 2: LITERATURE REVIEW

This chapter discusses related work along with concepts and required background of prior research relevant to this dissertation. First, it highlights access control preliminaries along with some important traditional models including the attribute based access control which is more flexible and suits current distributed multi-tenant applications. It also provides overview of some administrative models which inspire the GURA_G model developed in the dissertation along with earlier work done in safety and reachability analysis problems. Related work and technologies used in smart cars ecosystem along with security proposals made by several federal and other agencies is briefly discussed. Literature in solutions provided for Big Data security and Hadoop is also presented. The order of sections is based on their relevance in the chapters of dissertation.

2.1 Access Control Models

Access control is an important security mechanism that determines what operations (or actions) a user (or subject) can perform on which resources (or objects) [145]. Authentication determines identity of the user and answers *who* whereas authorization checks *what* can be done on *which* resources by the authenticated users. Several access control models have been proposed in literature and act as a primary cyber attack prevention technique for computer applications and systems.

Discretionary Access Control (DAC) [145], Mandatory Access Control (MAC) [145], and Role-Based Access Control (RBAC) [72,144] are three most significant and widely adopted access control models. In DAC, the discretion of the owners of objects determines the access of users to their objects. When a user requests access to an object, the request is checked against the defined policy for that object from the owner which determines either to grant or deny the access request. The cells in the access control matrix determine the operations which a subject (in rows) can have on other subjects or objects (in columns). Access control list (ACLs), capabilities and relations are some ways to implement DAC. This simple and easy to setup access control model has an inherent weakness of trojan horses where copying of information from one object to another cannot be

controlled and can be exploited to get unauthorized access of sensitive information.

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

RBAC or Role Based Access Control [72, 144], developed around mid 1990s, is an administrative friendly access control model. It determines accesses based on roles assigned to the users and permissions associated with these roles on specific objects with defined operation. In this approach, an administrator creates roles that represent specific tasks and assigns permissions to those roles (permission-role assignment), and these roles are then assigned to users (user-role assignment). RBAC is capable of enforcing both DAC and MAC models [133]. Major cloud platforms such as Amazon AWS [1], Microsoft Azure [14], and OpenStack [15] utilize role-based access control model as their authorization foundation. Apart from many known advantages of RBAC, it also has some limitations such as role explosion (where too many roles are created in the organization) and role-permission explosion (where too many permissions are assigned to roles and becomes hard to track). Further in RBAC, access control policies can only be defined on basis of roles which restricts access control flexibility. RBAC administration is also an extra overhead.

Some other access control models proposed include provenance based access control (PBAC) [135] and relationship based access control (ReBAC). PBAC uses the provenance information attached with the underlying data, which offers utilities as usage information, versioning or pedigree information, to provide access controls. It can further support dynamic separation of duties and workflow controls. ReBAC [56] enables permissions based on the relationship among users and

primarily used in online social networks (OSNs). Besides user to user (U2U) relationship, user to resource (U2R) and resource to resource (R2R) relation have also been used to control usage and administrative activities of the users in OSNs.

Lately, attribute based access control model (ABAC) has received significant attention where the authorization decision for operations by users on objects is based on the attributes (characteristics) of the users and objects in question. Authorization policies are defined using policy language which includes attributes that the users and objects should have to satisfy the policy and determine access control decision. Besides the attributes of users and objects other information like the contextual attributes, environment attributes like time, date, ip address or alert level can be used to fine grain the decision, to offer needed flexibility which is very important in distributed multi-tenant environment. The National Institute for Standards and Technology (NIST) has published a special publication [97] to provide detailed ABAC concepts, capabilities and implementation architecture. Jin et al [105] proposed a unified ABAC model called ABAC_α which can be configured to enforce DAC, MAC and RBAC models. Combinations of RBAC and ABAC models have also been proposed by adding attributes to role-based access control policies [114]. Dynamic roles uses user and context attributes for assigning roles to users dynamically, similar to attribute-based user-role assignment [36]. Attribute-centric approach assume roles as another attribute of users. Role-centric approach curtails the permissions of a role based on user attributes [108, 114]. Further, Yong et al [185] proposed extending the roles of the RBAC with attributes.

2.2 Administrative Models and Safety Analysis

Administrative models complement the operational access control models. In case of RBAC, the administration part of the model address the following questions: who will assign roles to the user, how permissions are added to the roles, who will assign these permissions, how the role hierarchy exists, and so on. Similarly, in ABAC administrative models the questions arise as to how the attributes are assigned to user, objects or other relevant entities, who will assign these attributes and any set of preconditions needed to have a particular attribute. All these questions need to be

resolved. Hereafter, this section highlights some important administrative models followed by an explanation of safety analysis problem and relevant work.

2.2.1 ARBAC97 and GURA Models

Administrative role based access control (ARBAC97) model [140] was proposed for RBAC administration. It has three components: URA (user-role assignment), PRA (permission-role assignment), and RRA (role-role assignment) dealing with different aspects of RBAC administration. In URA97, the goal is to impose restrictions on which users can be added to a role by which administrator, as well as to distinctly separate the ability to add and remove users from other operations on the role. Prerequisite conditions are important component of URA97 model and are boolean expressions using *and* and *or* operations on current roles of the user in question. PRA97 is concerned with role-permission assignment and revocation where the boolean expression is evaluated for membership and nonmembership of a permission in specified roles. The RRA97 model distinguishes roles into three mutually disjoint types: *Abilities* (A), *Groups* (G) and *UP-Roles* (UPR). An ability is a collection of permissions which should be assigned as a single unit to a role whereas a group is a collection of users who are assigned as a single unit to a role. UP-Roles have no restriction on membership, which can include users, groups, permissions, abilities, or other UP-roles as their member. PRA97 model can be used to propose a similar ARA97 model for ability-role assignment whereas the URA97 model can be used to produce the GRA97 model for group-role assignment. The administration and role hierarchy related to UP-roles is detailed in [140].

The generalized URA (GURA) model [106] was proposed to assign attributes to users. The model is composed of three relations defined by the security architect for adding set-valued user attribute, deleting set-valued user attribute and assigning atomic-valued user attribute. These relations are distinguished based on the attribute type, which can be set valued or atomic valued. A set valued attribute can have multiple values whereas an atomic attribute only has a single value. Each tuple in these relations define an administrative rule which include an administrative role which can assign or remove attribute values (chosen from the range of the allowed values) from the user

who satisfy a set of specified preconditions written using a policy language. One limitation of this model is the assignment of administrative role to the users, which itself needs administrative rules. Also distributed policies are hard to express in the policy language specified in the GURA model. Crampton and Loizou [63] also presented an administrative work related to RBAC model [72] and developed models for role hierarchy administration. The URA97 and GURA model provides inspiration for our proposed $GURA_G$ model as discussed in Chapter 3.

2.2.2 Safety and Reachability Analysis

Reachability analysis for user attributes is well studied in [107], which is based on GURA administrative model [106]. In this analysis, attributes are assigned to users directly depending on certain attribute based prerequisite conditions and on administrative roles. This work proves PSPACE-complete complexity for generalized GURA scheme and also presents polynomial algorithms for some conditional cases. Our work in Chapter 3 extends this reachability analysis as it involves attributes assignments to users as well as to groups to which users are members. This assignment of attributes to groups provides administrative benefits in adding and removing multiple attributes to users with single administrative operation.

Security policies have been widely analysed in several works including [94, 104, 120–122, 138, 141, 142, 148, 150, 166]. The safety analysis problem goes back to 1970's. In general, the safety of access control matrix (ACM) model has been shown to be undecidable [94]. Tripunitara and Li [166] presented an important theoretical comparison of expressive powers of different access control models. Some of the notations in Chapter 3 are from this work. Same authors also defined restricted forms of ARBAC97 (AATU and AAR) and provided algorithms for analysis problems including safety and availability in restricted forms [121]. Our work extends results from trust management policies [120] where safety and availability security analysis on delegation of authority is discussed. The schematic protection model (SPM) [141] introduced typed security entities where each entity is associated with a security type, which remains unchanged. Sasturkar et al [148] analyse ARBAC97 administrative policies to determine reachability and availability problems, by

establishing connections to the artificial intelligence planning problem. Jha et al [104] classified analysis problems related to RBAC and showed PSPACE-complete complexity for unrestricted classes whereas NP-complete and polynomial time algorithms exists for restricted subclasses. Lipton et al [122] presented a linear time algorithm for take and grant system. Recently, Rajkumar and Sandhu [138] discussed safety problem for pre-authorization sub-model for $\mathrm{UCON}_{\mathrm{ABC}}$.

Jajodia et al [103] presented a logical language to express positive, negative and derived authorization policies, and provided polynomial algorithms to check its completeness and consistency. Cholvy and Cuppens [57] discussed the problem of policy consistency and offer a methodology to solve it. They further suggested the use of roles priorities to resolve normative conflicts in policies. [45] provides a method to transform policy specifications into event calculus based formal notation. It further describes the use of abductive logical reasoning to perform a priori analysis of various policy specifications. Jaeger et al [102] presented the concept of access control space and its use in managing access control policies. These spaces are used to represent permission assignment to subjects or roles. Authors in [73] presented decision diagram based algorithms to analyze XACML based policies and compute the semantic differencing information between versions of policies. Alloy language is used for specification of role based system and analysis is done using Alloy constraint analyser [150]. Stoller et al [158] provided algorithms for ARBAC97 policies limited to rules with one positive precondition and unconditional role revocations. Same authors in [157] defined PARBAC (parameterized ARBAC) and determined user-reachability problem as undecidable over infinite types of parameter. They further assume all parameters are atomic-valued and are changed when the role is modified. Gupta et al [91] discussed rule-based administrative model to control addition and removal of facts (attributes) and rules. They further proposed an abductive algorithm which can analyze policies even when the facts (attributes) are unavailable based on computation of minimal sets of facts. The work in [109] provides analysis of expressive power of generalized temporal role-based access control (GTRBAC) which offers a set temporal constraints to specify fine grained time based policies.

2.3 Smart Connected Vehicles

Internet of Things (IoT) is the new era of technology which envisions to make human lives smarter. The concept has attracted wide applications and services in variety of domains including healthcare, homes, industry, transportation, power grids etc. The magnitude of this technology is illustrated by the number of devices which are estimated to be more than 20 billion by year 2020 [78]. The prime asset delivered by such massive interconnection and networking of smart devices is Big Data, which is analyzed to gather insights and deliver valuable information.

IoT requires the use of multiple technologies including identification (naming and addressing), sensing (sensor devices, RFID tags etc.), communication technologies (Bluetooth, WiFi etc.), computation technologies involving hardware or software platforms like Cloud, multiple IoT services [81] and the applications which provide functionalities to the end user [35, 41, 83]. Several IoT architectures have been demonstrated to incorporate physical objects, object abstraction (virtual objects), middleware or service, application and business layers with variations in architecture stack and nomenclature [35, 41]. Cloud computing is also an important domain in today's world which offers boundless applications and resources (storage and compute) to multiple users. Therefore, the merger of IoT and cloud is arguably indispensable to harness the full potential of IoT smart objects which have limited storage, processing and communication capabilities. The literature has recognized this desirable integration using terms such as cloud-assisted, cloud-enabled, and cloud-centric IoT [34, 48, 50, 52, 53, 65, 118].

Vehicular IoT and connected cars is a novel domain where networking and communication among cars, traffic infrastructure, pedestrians, homes or ultimately anything is proposed. The prime goal of vehicular IoT is inter-connectivity among smart entities in which vehicles are most important. As stated [23] in Wikipedia, "A Connected car (or Connected Vehicle (CV)) is a car equipped with internet access and usually also with the wireless area network. This allows the car to share internet access with other devices both inside as well as outside the vehicle". Gartner predicts a quarter billion connected cars by year 2020 [77] which will form a significant portion of the overall connected devices. The communication among vehicles and infrastructure, driving

assistance and autonomous driving, automatic braking and emergency calling, weather and accident warnings, parking areas, E-toll, and predictive maintenance, are among the most desired and available features in today's connected cars. These cars have more than 100 ECUs and 100 millions lines of code in support of such functionality. CVs have controller area network (CAN) bus with FlexRay, Ethernet and other protocols which are used for ECU communication within the car. Messages are broadcasted to all ECUs attached to bus. Multiple buses are connected via a gateway, usually a TCU (Telematics Control Unit), which also provides interface to external environment. These vehicles generate, exchange and process huge amounts of data and are often referred to as 'smartphones on wheels' [165]. As vehicles with a broad attack surface get connected to the internet, they get exposed to remote malicious activities. Cyber attacks can be orchestrated from in-vehicle network, from a user inside the car using a smartphone, from external entities in proximity, or even through cloud. Some of the most hackable and exposed attack surfaces in a connected car include airbag ECU, Bluetooth, TPMS, and remote key [54].

This emerging concept involves several new and established technologies which needs to be discussed to understand Internet of Vehicles (IoV) systems. This section reviews vehicular IoT and smart cars building blocks along with security concerns which we believe are fundamentally required and are the basis of our research work in Chapter 4.

2.3.1 VANETs and Vehicular Cloud

Vehicular Ad-hoc Networks (VANETs) have been proposed in the literature to support vehicle to vehicle and vehicle to infrastructure communication which enable advanced services to the drivers. The network nodes in VANETs (cars, infrastructure etc.) have storage, computation and communication modules to provide such services. However, most of these on-board resources are usually under-utilized with the set of applications offered, and can be utilized for additional services to stake holders [69, 131]. The concept of vehicular cloud (VC) has been proposed which blends the two separate ideas of VANETs and Cloud computing. Cloud computing provides the idea of boundless storage, compute or network resources in the form of Infrastructure as a Service

(IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS), which are extended to the inter-networked cars and infrastructure provided by VANETs. Vehicular Cloud [69, 79, 80, 131] utilizes coordinated on-board resources of cars and infrastructure to offer the capabilities of 'cloud on the fly' to users that need them.

The vision of intelligent transportation system (ITS) requires cooperation among entities for smooth and efficient traffic flow with information and entertainment (infotainment) to driver. All such applications have local relevance which need time and location sensitive computation of information avoiding the latency and bandwidth problems when the information is loaded and processed in central cloud. Therefore, the surrounding vehicles can form autonomous clouds to solve driver's locally relevant queries about traffic ahead or parking nearby. Several architectures have been proposed for the formation of vehicular clouds like stationary VC, VC linked with a fixed infrastructure or dynamic VC, where each has different formation scenarios [101, 180]. The key features to distinguish conventional cloud and VC are mobility, agility and autonomy of vehicles, which are computation and storage nodes in vehicular cloud. In VCs, one vehicle is selected as the broker by surrounding vehicles which mediates resource sharing among vehicles in and around specific geographic boundary (for example in 2 miles radius). The broker asks permission for cloud formation from relevant authorities and also sends request to neighbouring vehicles to share resources. Once approved by authorities (DMV or transportation agency), these vehicles pool their resources to form a virtual environment which is shared by all VC users. Further, large scale federation of VCs can be established in case of emergency situations like earthquake, providing temporary infrastructure when conventional cloud is unreachable.

Our research assumes that vehicular IoT will involve single or multiple cloud/fog instances supporting different service models – SaaS, PaaS and IaaS. These instances can cover wide geographic area using central cloud, fog instances within 1-2 miles radius or even fog instances at each connected car level based on different use cases. These architectures can be public, private (for example by a car manufacturer) or hybrid and involve single internet clouds, vehicular clouds, fog instances or any combination of them as discussed further in Chapter 4.

2.3.2 Virtual Objects in IoT

The cyber-physical ecosystem of vehicular IoT has heterogenous objects with different operating conditions, communication technologies, and functionalities. Further, the issues related to object connectivity, scalability, object and service discovery, security and privacy, quality management, and identification are challenges in any IoT system [129]. To counter these issues the concept of virtual objects is introduced in several IoT architectures [147, 179]. Amazon AWS IoT [49] also incorporates virtual objects as device shadows where in case a physical device is not connected, its cyber counterpart (i.e. shadow) will have the last received state or desired future state information. Therefore, whenever the physical device gets connected to its virtual entity, it gets updated to the state of its cyber object and also mitigates the problem of sporadic object connectivity. Microsoft Azure [32] has device twins which are JSON documents maintained in Azure IoT hub for each device connected and stores device state information. Different association scenarios exist between physical and virtual objects: single virtual object for one physical object irrespective of the number of services and functionalities provided by physical object; whereas for object with multiple services, it is possible to have many virtual objects for each service of same physical object. Similarly, other configurations such as many physical to one virtual or many physical to many virtual mappings are also possible depending on different use-case requirements. The creation and location of virtual objects is primarily proposed in the cloud and their communication uses RESTful technologies [129]. Since high latency and low bandwidth issues will exist in virtual objects creation, for real time applications like vehicular IoT, this dissertation envisions to keep the virtual objects near to the physical objects, i.e at the fog level or in vehicular cloud (VC).

2.3.3 Access Control Oriented Architecture

Several IoT architectures have been proposed in the literature [35,41,52,83,129] and in general, all such architectures have three layers: object, middleware (with multiple sub-layers) and application layer. Recently, Alsehri and Sandhu proposed an IoT architecture, referred as access control oriented architecture (ACO) [37], taking into consideration the access control requirements in IoT and

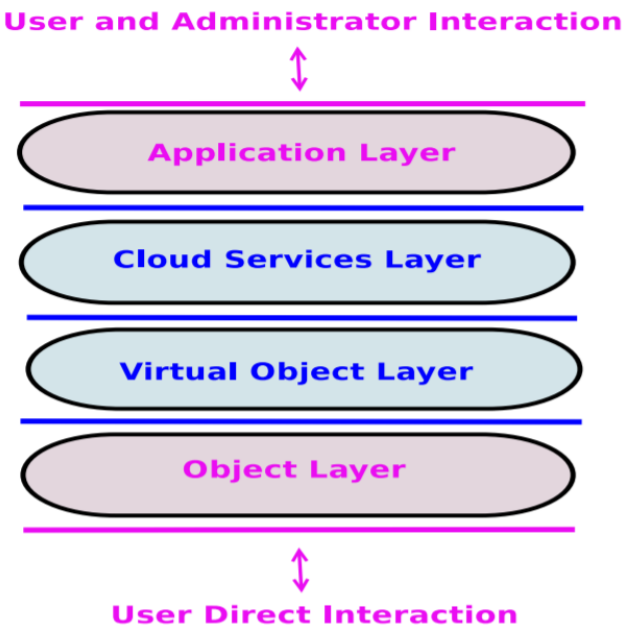


Figure 2.1: ACO Architecture [37]

incorporation of different models at various layers. As shown in Figure 2.1, ACO architecture has four layers – object, virtual object, cloud services and application – with user and administrators interacting at both object and application layers. Since, our proposed extended ACO architecture for connected cars ecosystem (discussed in Chapter 4) adds to/refines generic IoT based ACO architecture, we briefly outline ACO architecture layers below.

- **Object Layer:** The bottom layer of ACO architecture comprises physical smart devices like sensors, RFIDs, beacons, and ECUs, which are responsible for data sensing and accumulation, and for sending data to upper layers. These devices can communicate with other devices using different communication technologies including Bluetooth, WiFi, Zigbee, LAN and LTE. Physical devices communicate with their cyber counterparts (virtual objects) using protocols like HTTP, MQTT, DDS or CoAP [35]. Users can also directly access physical objects at this layer which generally have limited computational power and low storage

- **Virtual Object Layer:** As discussed earlier, virtual objects represent the digital counterpart of physical objects which maintain the status of physical objects even when they are not connected. ACO architecture recommends virtual object layer as a part of middleware to support

communication between heterogenous objects and overcome IoT challenges of scalability or locality. In ACO architecture only objects are assumed to have virtual counterparts.

- **Cloud Services Layer:** With the number of IoT devices proliferating, the storage and computation of data will be done in cloud, where different applications can harness it to make valued decisions. Single or multiple cloud scenarios can exist to support federation or trusted collaboration between them. Some important IoT cloud platforms include Amazon AWS [42], Microsoft Azure IoT Hub [32], and Google Cloud IoT Core [30].

- **Application Layer:** The applications offered by IoT systems to end users are situated in this layer, which leverage the services and functionalities of the lower cloud services layer. Users and administrators can remotely send commands and instructions to smart devices at bottom layer using these applications, but such interaction has to pass via other two ACO middleware layers (cloud services and virtual object). Administrators can also define access control policies for various IoT resources using this layer.

2.3.4 Cyber Security Concerns and Proposed Solutions

Connected vehicle with more than 100 ECUs, with broad attack surface interacting both in-vehicle systems and a wide range of external networks including WiFi, cellular networks, and internet to data exchange between service garages, toll roads, gas stations, and several automotive and aftermarket applications [54], present a big challenge for security. Most security vulnerabilities like trojan horse, buffer overflow exploits, malware, ransomware, and privilege escalation can be exploited on connected vehicles and other smart transportation entities. Recently Tesla Model X was hacked [172]. Figure 2.2 shows fifteen most hackable and exposed attack surfaces in the connected car as discussed in [54]. Security is vital in smart cars ecosystem where attacks (like disabling brakes) can even lead to loss of life. Several studies and reports [17, 24, 136, 139, 175] have been published to illustrate potential risks and attacks which can be orchestrated on smart entities in IoV. Some examples of cyber attacks in connected cars and IoV as discussed in [61, 68, 76, 100, 159] include: user impersonation to exchange fake basic safety messages (BSM) or

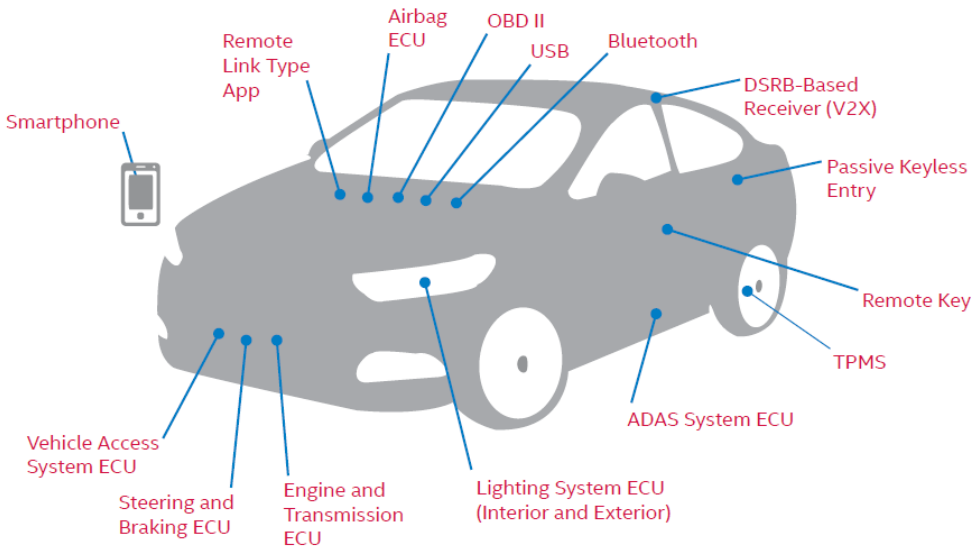


Figure 2.2: Most Exploitable Attack Surfaces [54]

false information about an accident, stealing personal data or credit card information, controlling critical sensors of connected vehicle, gaining knowledge of vehicle and driver movement, spoofing CV's sensors, coordinated attacks on infrastructure, unauthorized over-the air firmware updates, and infecting a CV with ransomware. CAN bus used for internal ECU communication must also be secured to prevent unauthorized gain of data and manipulation of software on ECU and sensor systems. An unauthorized party that gains access to the bus can block legitimate messages and transmit illegitimate ones. On board equipments (OBEs) integrate with the CAN bus to provide information such as vehicle speed and brake system status to participating entities. This bring us back full circle to needing to protect the internal components of a vehicle in order to maintain confidence that V2V, V2I and V2X messages are legitimate. Securing IoV and connected vehicles will require protecting control systems (on-board diagnostic (OBD) port, CAN bus etc.), protecting infotainment systems, securing smartphone applications, securing infrastructure, securing over-the air updates, and securing hardware from manual tampering. Security mechanisms become hard to implement considering intrinsic vehicular IoT characteristics like dynamic topology, mobile limitation, and large scale network.

US Department of Transportation initiated the ITS (Intelligent Transportation Systems) program to enable communication among vehicles and other smart infrastructures while ensuring

security and privacy of the stake-holders. The BSMs exchanged among entities must not include personally identifiable information and must be broadcast in limited geographic area [173]. Dedicated short range communications (DSRC) is used to exchange information across entities which is used by several safety and other applications to generate alerts for drivers. Therefore, the confidentiality and integrity of such messages is imperative so that drivers can trust their source and information in them. Security Credential Management System (SCMS) [174] has been proposed to ensure trust and message security using public key infrastructure (PKI) approach where certificate generated by certificate authority (CA) is attached with the BSM to ensure trust between talking entities. European Union Agency for Network and Information Security (ENISA) has also released a study in year 2017 [70] which enumerates critical assets in smart cars, threats, potential risks, and proposes good practices mainly segmented into three categories, policy and standards, organizational measures, and security functions, to ensure security of smart cars against cyber threats. European Commission has set up Cooperative Intelligent Transport Systems (C-ITS) Deployment Platform to foster cooperative, connected and automated vehicles, and has released security frameworks [171] and certificate policy [170] documents. National Institute for Standards and Technology (NIST) also proposed a framework [128] for cyber-physical systems (CPS) which address conceptualization, realization and assurance of CPS including security and interoperability.

2.4 Big Data and Hadoop Ecosystem

In past several years, enterprises have grown their reliance on Big Data for critical financial and strategic decisions. An estimate in IDC's Digital Universe study, predicts world's data size to reach 44 zettabytes by 2020 [75]. Multi-format big data is collected from diverse sources including sensors, smart connected cars, tennis rackets, web browsing, social media, power meters etc., to improve organization's operational efficiency, revenue and to offer personalised customer experience. Leveraging full potential and gaining valuable insights of such massive data sets require enormous infrastructure for storage and computation in real time manner. This section provides literature review for our work in big data security in Hadoop ecosystem discussed in Chapter 5.

Apache Hadoop [3] offers a distributed, scalable and cost-efficient open-source framework for storing and analysing structured, unstructured and semi-structured data in variety of formats. Resilient storage and rich analytical capabilities provided by Hadoop and its ecosystem components (Apache HBase, Apache Hive etc.) makes it a prime choice as a big data processing system in government and industry. Such wide acceptability of Hadoop ecosystem comes with the responsibility to make it secure against cyber attacks.

The Multi-tenant Hadoop Data lake stores sensitive information including credit card numbers, medical records and social security numbers (SSNs), requiring the cluster to be protected against cyber threats. Unauthorized access to data assets can have serious impact on its confidentiality and integrity. An inside user can masquerade by running malicious code to impersonate Hadoop core services including HDFS NameNode, DataNode or YARN ResourceManager. A nefarious user can also modify, view or delete other users' applications. It is also possible to execute denial of resources attack, where a malicious user can submit lengthy jobs which consume all the cluster resources preventing other users from submitting new jobs. The challenges to mitigate these threats include distributed and partitioned file system and computing, scale of Hadoop cluster, multi-tenant environment and multi-level access of same data elements to different users. Correspondingly, Hadoop ecosystem has deployed several security measures including authentication, authorization, data encryption and network security.

Access Control mechanisms are vital in restricting users and applications access to authorized resources. Apache Hadoop deploys a multi-layer authorization framework using Access Control Lists (ACLs) to authorize users to access data, infrastructure resources and services in Hadoop cluster. Apache Ranger [7] and Apache Sentry [8] are two widely deployed systems to enforce finer grained authorization across several Hadoop ecosystem services. Both systems offer a centralized administration console to store and manage security policies for multiple ecosystem components. They provide plugins which are hooked to ecosystem services to decide and enforce access control, based on the policies pulled from central policy server. Sentry supports role-based authorization model, whereas Ranger assigns permissions to users and groups.

Several papers [18, 20, 64, 84, 132, 154, 181] discuss security threats and solutions in Hadoop ecosystem. Recently, Gupta et al [84] presented a multi-layer authorization framework for Hadoop ecosystem, which covers several access control enforcement points and demonstrates their application using Apache Ranger. Access control using cryptography based on proxy re-encryption [130] provides approach for delegated access to Hadoop cluster. A security model for G-Hadoop framework using public key and SSL is presented in [189]. Security and privacy concerns of MapReduce applications are discussed in [66]. Ulusoy et al [168, 169] proposed approaches for fine grained access control for MapReduce systems. Privacy issues in Big Data are addressed in [60, 124, 156, 163]. Risk aware information disclosure in [40] can be used for Hadoop Data lake. Secure information access model via data services [47] can be applied for Hadoop data services. HDFS can use data access protection using data distribution and swapping in [67]. Vimercati et al [176] discuss confidentiality of outsourced data. Colombo et al [59] also proposed fine-grained context-aware access control features for MongoDB NoSQL datastore.

Risk based access using classification [44] studies role assignment based on risk factors. Contextual attributes in location aware ABAC in [96] can be applied in Hadoop. Classification of data object based on content is presented in [183]. Policy engineering for ABAC [113] can be used to define values based on risk or context. Another promising approach in attribute based data sharing has been presented in [186]. Use of role mining in [123] can be extended to determine roles of users based on attributes. A research roadmap on trust and Big Data is presented in [146]. Trust based Data ingestion or processing can use models in [125]. Hu et al [98] presented a general access control model for big data processing frameworks. The paper introduces chain of trust among several entities to authorize access request. The work provides a preliminary document which can be conceptualized to specific systems like Hadoop. However, the authors do not address details particular to the Hadoop ecosystem.

CHAPTER 3: GROUP BASED ATTRIBUTES ADMINISTRATION IN ABAC AND REACHABILITY ANALYSIS

In this chapter, we propose an administration model for group and user attributes assignment in ABAC, referred as GURA_G. We also study the user attribute reachability analysis based on user to groups memberships and the administrative rules stated using policy language defined in GURA_G model. Significant portion of this chapter has been published at the following venue [88].

- Maanak Gupta and Ravi Sandhu, The GURA_G Administrative Model for User and Group Attribute Assignment. In Proceedings of the 10th International Conference on Network and System Security (NSS), September 28-30, 2016, Taipei, Taiwan, pages 318-332.

3.1 Motivation

Attribute based access control determine the permissions based on the attributes of the users and objects in access request. Hierarchical Group and Attribute based access control (HGABAC) [153] model was recently proposed which introduced the novel notion of attribute inheritance through groups memberships. Besides, the users and objects directly assigned attributes, they also inherit attributes from groups memberships, which offers administrative benefits since multiple attributes can be assigned to user or objects by a single administrative operation. The GURA_G model provides an administration model for user attributes in HGABAC model by defining the rules which determine the preconditions under which users are assigned or removed attributes based on the changing group memberships by administrators. Also since the attributes of entities define their permissions, it is important to understand the set of attributes an entity can attain directly or inherited via groups memberships with a predefined set of administrative rules. Therefore, we also study the reachability analysis to determine the effective user attributes based on member groups attributes and the administrative rules defined under GURA_G model. We claim that in general the problem is PSPACE-complete, however, under restricted pre-conditions we provide polynomial time algorithms for different restricted GURA_G instances.

3.2 Hierarchical Group and Attribute Based Access Control

This section gives an informal characterization of groups in HGABAC [153], followed by a formal specification. Our formalization is in the style of ABAC_α [105], different from but equivalent to the formalization of Servos et al [153]. Our alternate formalization of HGABAC enables us to build GURA_G model upon the GURA administrative model [106] for ABAC_α in Section 3.3

3.2.1 Groups in HGABAC

Similar to many ABAC models, HGABAC recognizes the entities of users, subjects and objects. A user is a human being which interacts directly with the computer, while subjects are active entities (like processes) created by the user to perform actions on objects. Objects are system resources like files, applications etc. Operations correspond to access modes (e.g. read, write) provided by the system and can be exercised by a subject on an object. The properties of entities in the system are reflected using attributes. Users and subjects hold the same set of attributes whereas objects have a separate set of attributes reflecting their characteristics. All attributes are assumed to be set valued where each attribute has a finite set of possible atomic values from which a subset can be assigned to appropriate entities. In addition to these familiar ABAC entities, HGABAC further introduces the notion of a group as a named collection of users or objects. Each group has attribute values assigned to it. A member of the group inherits these values from the group. Users will inherit attributes from user groups and objects from object groups. A partially ordered group hierarchy also exists in the system where senior groups inherit attribute values from junior groups.

An example user-group hierarchy is illustrated in Figure 3.1. Senior groups are shown higher up and the arrows indicate the direction of attribute inheritance. Since Graduate group (G) is senior to both CSD and UN, G will hold the attribute values directly assigned to it as well as values inherited from CSD and UN. The values of univId and college attributes for group G are respectively inherited from UN and CSD, values of userType and studType are directly assigned to G while the values of roomAcc are a mix of directly assigned values, 2.03 and 2.04, and inherited value 3.02 from CSD. Each user is assigned to a subset of user groups. Similarly there is an

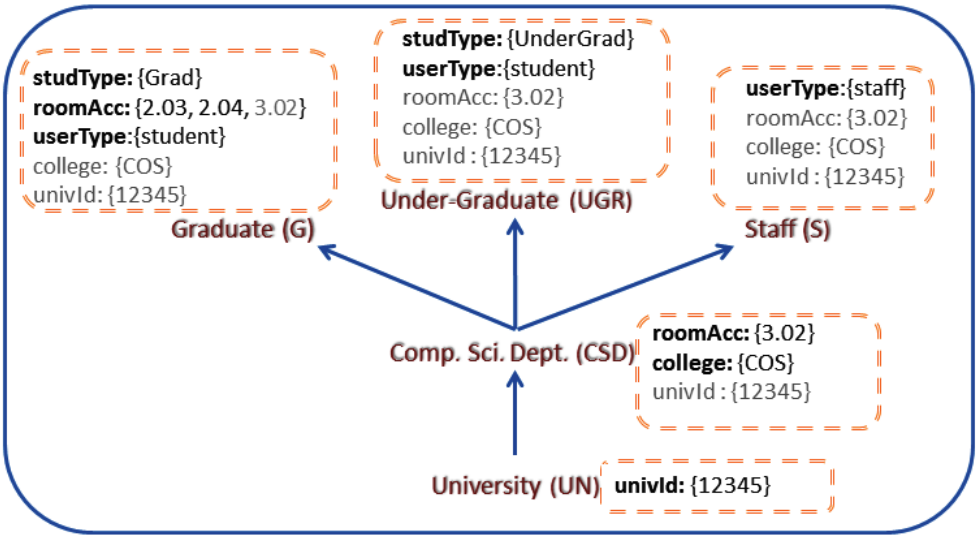


Figure 3.1: Example User Groups (values in black are direct and in gray are inherited)

object-group hierarchy wherein attribute values of objects are analogously inherited.

The core advantage of introducing groups is simplified administration of user and object attributes where an entity obtains a set of attributes values by group membership in lieu of assigning one value at a time. In context of Figure 3.1 assigning an attribute value to CSD potentially saves hundreds or thousands of assignments to individual student and staff. Likewise changing the CSD level room from 3.02 to, say, 3.08, requires only one update as opposed to thousands.

3.2.2 HGABAC: An Alternate Formalization

We now develop a formalization of the HGABAC model different from that of Servos et al [153]. This alternate formalization will be useful in the next section where we develop the $\mathrm{GURA_G}$ for administration of HGABAC. Our formalization uses the conceptual model of HGABAC shown in Figure 3.2. The complete HGABAC formalization is given in Table 3.1, which we will discuss in the remainder of this subsection followed by an example configuration.

Basic sets and functions of HGABAC are shown at the top of Table 3.1. U, S, O and OP represent the finite set of existing users, subjects, objects and operations respectively. UG and OG represent sets of user and object groups in the system. UA is the set of user attributes for users, user groups and subjects. OA is similarly the set of object attributes for objects and object groups.

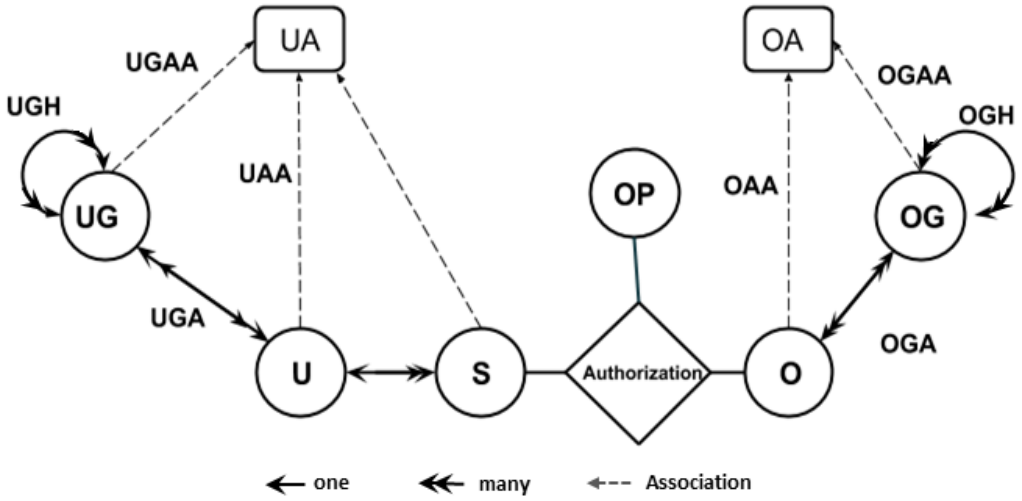


Figure 3.2: A Conceptual Model of HGABAC

All these sets are disjoint and are predefined in the system.

Attribute values can be directly assigned to users, objects, user groups and object groups (we will consider subjects in a moment). These are collectively called entities. Each attribute of an entity is set valued. The value of an attribute att for an entity is some subset of Range(att) which is a finite set of atomic values, as indicated by the functions $\mathrm{att_u}$ and $\mathrm{att_o}$ in Table 3.1. These functions specify the attribute values that are directly assigned to entities. The function directUg specifies the user groups to which the user is assigned, and similarly the function directOg specifies the object groups to which an object is assigned. User group hierarchy (UGH) is a partial order on UG, written as $\succeq_{ug}$, where $ug_1 \succeq_{ug} ug_2$ denotes ug_1 is senior to ug_2 or ug_2 is junior to ug_1. This many to many hierarchy results in attribute inheritance where the effective values of user attribute function $\mathrm{att_u}$ for a user-group *ug* (defined by $\mathrm{effectiveUG_{att_u}}(ug)$) is the union of directly assigned values for $\mathrm{att_u}$ and the effective attribute values of all groups junior to *ug*. The assignment of a user to a user-group will inherit values from this group to that user. The function $\mathrm{effective_{att_u}}$ maps a user to the set of values which is the union of the values of $\mathrm{att_u}$ directly assigned to the user and the effective values of attribute $\mathrm{att_u}$ from all user groups directly assigned to the user. Similar sets and functions are specified for objects and object groups.

A subject is created by a user, denoted by the SubUser function. The effective attribute values

Table 3.1: HGABAC: An Alternate Formal Model

Basic Sets and Functions

– U, S, O, OP (finite set of users, subjects, objects and operations respectively)
– UG, OG (finite set of user and object groups respectively)
– UA, OA (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, $\mathrm{att_u} : \mathrm{U} \cup \mathrm{UG} \rightarrow 2^{\mathrm{Range(att_u)}}$, mapping each user and user group to a set of values in $\mathrm{Range(att_u)}$
– For each $\mathrm{att_o}$ in OA, $\mathrm{att_o} : \mathrm{O} \cup \mathrm{OG} \rightarrow 2^{\mathrm{Range(att_o)}}$, mapping each object and object group to a set of values in $\mathrm{Range(att_o)}$
– 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,
- $\mathrm{effectiveUG_{att_u}} : \mathrm{UG} \rightarrow 2^{\mathrm{Range(att_u)}}$,
  defined as $\mathrm{effectiveUG_{att_u}(ug_i) = att_u(ug_i) \cup (\bigcup_{\forall g \in \{ug_j | ug_i \succeq_{ug} ug_j\}} effectiveUG_{att_u}(g))}$
- $\mathrm{effective_{att_u}} : \mathrm{U} \rightarrow 2^{\mathrm{Range(att_u)}}$,
  defined as $\mathrm{effective_{att_u}(u) = att_u(u) \cup (\bigcup_{\forall g \in directUg(u)} effectiveUG_{att_u}(g))}$

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

Effective Attributes of Subjects (Assigned by Creator)

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

Authorization Function

For each op $\in$ 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:

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\, \mathrm{x} \in \mathrm{set}.\alpha \mid \forall\, \mathrm{x} \in \mathrm{set}.\alpha \mid \mathrm{set} \triangle \mathrm{set} \mid \mathrm{atomic} \in \mathrm{set} \mid \mathrm{atomic} \notin \mathrm{set}$
- $\triangle ::= \subset \mid \subseteq \mid \nsubseteq \mid \cap \mid \cup$
- $\mathrm{set} ::= \mathrm{effective_{att_{u_i}}(s)} \mid \mathrm{effective_{att_{o_i}}(o)}$ 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 $\in$ OP on a given object $\mathrm{o_j} \in$ O if the effective attributes of the subject and object satisfy the policies stated in $\mathrm{Authorization_{op}(s : S, o : O)}$. Formally, $\mathrm{Authorization_{op}(s_i, o_j)}$ = True

Table 3.2: Example HGABAC Configuration

Basic Sets and functions

– UA = {studId, userType, skills, studType, univId, roomAcc, college, jobTitle, studStatus}
– OA = {readerType}
– OP = {read}
– UG = {UN, CSD, G, UGR, S}, OG = { }
– UGH is given in Figure 3.1, OGH = { }
– Range of each $\mathrm{att_u}$ in UA, denoted by Range($\mathrm{att_u}$):
studId = {er35, abc12, fhu53}, userType = {faculty, staff, student},
skills = {c, c++, java}, studType = {Grad, UnderGrad},
univId = {12345}, roomAcc = {1.2, 2.03, 2.04, 3.02},
college = {COS, COE, BUS}, jobTitle = {TA, Grader, Admin},
studStatus = {graduated, part-time, full-time}
– Range of each $\mathrm{att_o}$ in OA, Range(readerType) = {faculty, staff, student}

Authorization Function:

$$\mathrm{Authorization_{read}(s : S, o : O)} \equiv$$
$$\mathrm{effective_{userType}(s)} \in \mathrm{effective_{readerType}(o)} \wedge \text{java} \in \mathrm{effective_{skills}(s)}$$

of a subject are under control of its creating user. These values are required to be a subset of the corresponding effective attribute values for the creator. In general these values can change with time but cannot exceed the creator's effective values. The exact manner in which a subject's effective attributes are modified by its creator is not specified in the model, and can be realized differently in various implementations.

Each operation op $\in$ OP in the system has an associated boolean authorization function $\mathrm{Authorization_{op}}$(s,o) which specifies the conditions under which subject s $\in$ S can execute operation op on object o $\in$ O. The condition is specified as a propositional logic formula using the policy language given in Table 3.1. This formula can only use the effective attribute values of the subject and object in question. The authorization functions are specified by the security policy architects when the system is created. Thereafter, a subject $\mathrm{s_i} \in$ S is allowed to execute operation op on object o $\mathrm{o_j} \in$ O if and only if $\mathrm{Authorization_{op}(s_i, o_j)}$ evaluates to True.

An example HGABAC configuration is given in Table 3.2, utilizing the user group hierarchy of Figure 3.1. For simplicity, we do not include any object groups. The authorization policy for the read operation is specified. The access request flow in Figure 3.3 assumes the user has the set

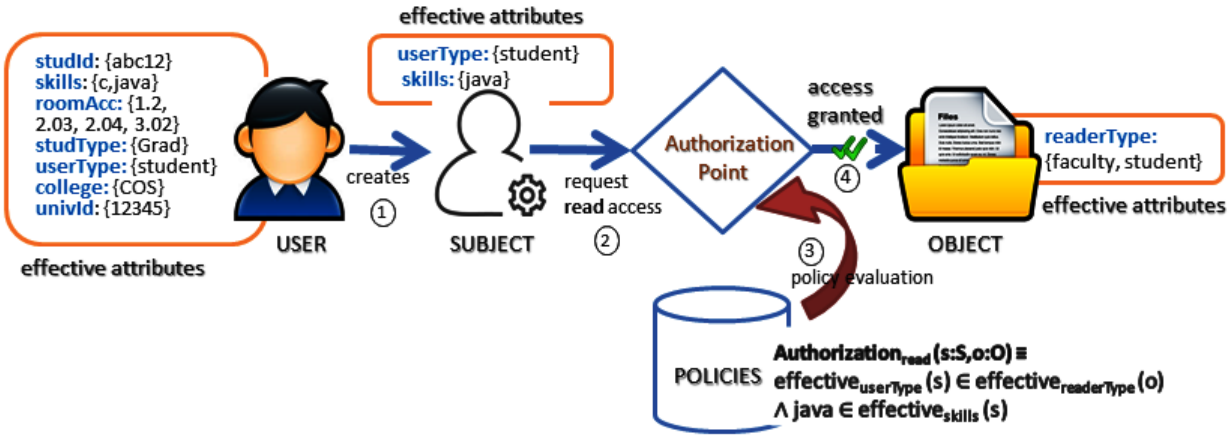


Figure 3.3: Example Access Request Flow

of effective attributes shown. The subject has the given subset of its creator's effective attributes. The subject is thereby allowed to read the object as the authorization policy for read is satisfied by the effective attributes of the subject and object.

3.3 The GURA$_G$ Administrative Model

The HGABAC model offers the advantage of easy administration of attributes for users and objects. The novel approach of assigning attributes to groups and users to groups is analogous to the permission-role and user-role assignment in RBAC [144]. By assigning a user to a user-group, the user inherits all the effective attribute values of that group in a single step, as compared to one by one attribute value assignment. Further, if an inherited attribute value has to be changed for multiple users, instead of changing per user, the value in a group can be changed, making administration very convenient. The essence and importance of HGABAC model is in simple administration as the effect of attribute inheritance can also be realized by direct attribute assignment for authorization purposes. Changing the attribute values of a group can impact large numbers of users and objects, thus reducing the administrative effort, and leading to better comprehension of attribute values. For example, in Figure 3.1 the fact that groups G, UGR and S inherit the roomAcc value 3.02 from CSD is visible because of the group structure.

This section presents the GURA$_G$ administrative model for managing the user side of

Table 3.3: $\mathrm{GURA_G}$ Administrative Model

Administrative Roles and Expressions

– AR : a finite set of administrative roles
– EXPR(UA) : a finite set of prerequisite expressions composed of user attribute functions as defined in Section 3.3.1 and 3.3.2
– EXPR(UA ∪ UG) : a finite set of prerequisite expressions composed of user attribute functions and user groups as defined in Section 3.3.3

Administrative Relations

– User Attribute Assignment **(UAA)** & User-Group Attribute Assignment **(UGAA):** For each $\mathrm{att_u}$ in UA,

$$\mathrm{canAdd_{att_u}} \subseteq \mathrm{AR} \times \mathrm{EXPR(UA)} \times 2^{\mathrm{Range(att_u)}}$$
$$\mathrm{canDelete_{att_u}} \subseteq \mathrm{AR} \times \mathrm{EXPR(UA)} \times 2^{\mathrm{Range(att_u)}}$$

– User to User-Group Assignment **(UGA):**

$$\mathrm{canAssign} \subseteq \mathrm{AR} \times \mathrm{EXPR(UA \cup UG)} \times 2^{\mathrm{UG}}$$
$$\mathrm{canRemove} \subseteq \mathrm{AR} \times \mathrm{EXPR(UA \cup UG)} \times 2^{\mathrm{UG}}$$

HGABAC. $\mathrm{GURA_G}$ is inspired by the GURA model [106] which in turn evolved from URA97 [140]. All these models require a set of administrative roles AR that will be assigned to security administrators. Administrative role hierarchy also exists, wherein senior administrative roles inherit permissions from junior ones. $\mathrm{GURA_G}$ regulates the powers of an administrative role with respect to user attribute assignment (UAA), user-group attribute assignment (UGAA) and user to user-group assignment (UGA) (see Figure 3.2). The *Add* and *Delete* operations enable addition or deletion of attribute values from user and user groups. Assignment or removal of a user from a user-group is accomplished by *Assign* and *Remove* operations. Table 3.3 depicts the various sets and administrative relations required to administer the user side of HGABAC. The prerequisite conditions are specified with slight modifications to the policy language described in Table 3.1. We now define the three sub-models of $\mathrm{GURA_G}$.

3.3.1 User Attribute Assignment (UAA) Sub-Model

The UAA sub-model deals with addition or deletion of values to a set-valued attribute of a user. It is composed of two relations as shown in Table 3.3. The meaning of $(\mathrm{ar}, \mathrm{Expr(ua)}, \mathrm{Z}) \in \mathrm{canAdd_{att_u}}$

Table 3.4: Example Administrative Rules in UAA

$\mathbf{canAdd_{jobTitle}}$ **rule :**
(DeptAdmin, Grad $\in \mathrm{effective_{studType}(u)}$, {TA, Grader})
$\mathbf{canDelete_{roomAcc}}$ **rule :**
(BuildAdmin, graduated $\in \mathrm{effective_{studStatus}(u)}$, {1.2, 2.03, 2.04, 3.02})

is that a member of an administrator role ar (or senior to ar) is authorized to add any value in the allowed range Z of attribute $\mathrm{att_u}$ of a user whose attributes satisfy the condition specified in $\mathrm{Expr(ua)}$. EXPR(UA) is the set of all prerequisite conditions represented as propositional logic expressions. The expressions return true or false and are specified using earlier defined policy language (Table 3.1) with following changes.

$$\mathrm{set} ::= \mathrm{att_{u_i}(u)} \mid \mathrm{effective_{att_{u_i}}(u)} \mid \mathrm{constantSet} \qquad \text{for } \mathrm{att_{u_i}} \in \text{UA}$$

$$\mathrm{atomic} ::= \mathrm{constantAtomic}$$

The meaning of $(\mathrm{ar}, \mathrm{Expr(ua)}, \mathrm{Z}) \in \mathrm{canDelete_{att_u}}$ is that the member of administrator role ar (or senior) is authorized to delete any value in allowed range Z of attribute $\mathrm{att_u}$ of a user whose attributes satisfy the condition specified in $\mathrm{Expr(ua)}$. The delete operation will only impact directly assigned attribute value of the user (i.e. $val \in \mathrm{att_u}(u)$). If the value to be deleted is inherited from a group, the operation will not have any effect. Further, if a value is both inherited and directly assigned to user, deletion will only delete the direct value, thereby, the user will still hold the value inherited from the group. It is worth mentioning that any change in prerequisite conditions after the attribute value assignment has been made, will not have any retrospective effect and the entity involved will still retain the value. This is consistent with the GURA and URA97 models.

Table 3.4 illustrates example UAA relation. First rule allows administrator role DeptAdmin (or senior to DeptAdmin) to add any value in {TA, Grader} to user attribute jobTitle if the user's effective studType attribute includes Grad. Second rule allows administrator role BuildAdmin (or senior to BuildAdmin) to remove any of the specified room values from the roomAcc attribute of a user whose effective studStatus attribute includes graduated value.

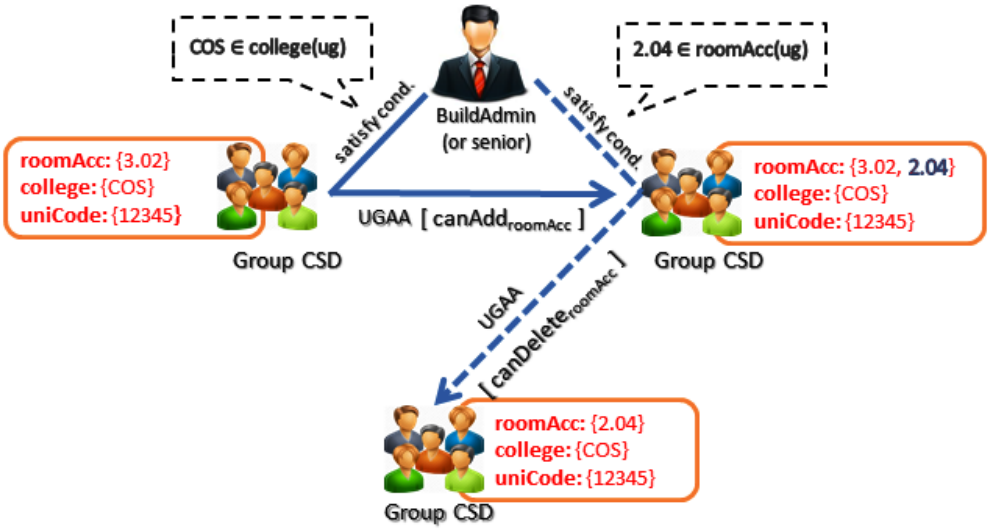


Figure 3.4: Example User-Group Attribute Assignment (UGAA)

Table 3.5: Example Administrative Rules in UGAA

$\mathbf{canAdd_{roomAcc}}$ **rule:** (BuildAdmin, COS $\in$ college(ug), {2.04})
$\mathbf{canAdd_{skills}}$ **rule:** (DeptAdmin, Grad $\in$ studType(ug), {c++})
$\mathbf{canDelete_{roomAcc}}$ **rule:** (BuildAdmin, 2.04 $\in$ roomAcc(ug), {3.02})

3.3.2 User Group Attribute Assignment (UGAA) Sub-Model

This sub-model controls addition and deletion of attributes from user-groups as shown in Table 3.3. The relations for UAA and UGAA have slightly different policy languages for EXPR(UA), which in case for UGAA is defined as follows.

$$\text{set} ::= \text{att}_{\text{u}_\text{i}}(\text{ug}) \mid \text{effectiveUG}_{\text{att}_{\text{u}_\text{i}}}(\text{ug}) \mid \text{constantSet} \qquad \text{for att}_{\text{u}_\text{i}} \in \text{UA}$$

$$\text{atomic} ::= \text{constantAtomic}$$

The meaning of canAdd and canDelete are similar to those in UAA sub-model. In particular, the delete operation in UGAA only impacts directly assigned attribute values of a user-group (i.e *val* $\in \text{att}_\text{u}(ug)$) and will not delete inherited values from junior groups.

Figure 3.4 shows addition and deletion of attribute values to user-group CSD in context of Table 3.5. Addition of value 2.04 to roomAcc attribute of CSD group by administrator role BuildAdmin (or senior to BuildAdmin) is allowed by first rule in Table 3.5. Figure also shows deletion of 3.02

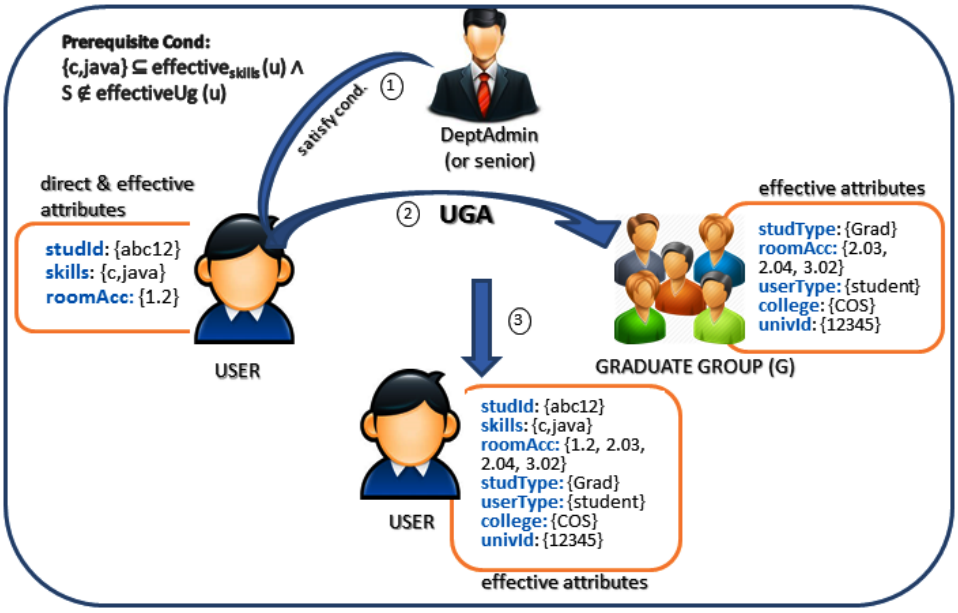


Figure 3.5: Example User to User-Group Assignment (UGA)

Table 3.6: Example Administrative Rules in canAssign UGA

Admin Role	Prereq. Cond	AllowedGroups
DeptAdmin	$\{c, java\} \subseteq effective_{skills}(u) \wedge$ $S \notin effectiveUg(u)$	{G,CSD}
StaffAdmin	$\{G,UGR\} \cap effectiveUg(u) = \ \wedge$ $Admin \in effective_{jobTitle}(u)$	{S}
DeptAdmin	$U \in directUg(u) \wedge$ $3.02 \in roomAcc(u) \wedge S \notin effectiveUg(u)$	{UGR,CSD}

value from roomAcc attribute authorized by third $\mathbf{canDelete_{roomAcc}}$ rule.

3.3.3 User to User-Group Assignment (UGA) Sub-Model

The UGA sub-model is composed of two authorization relations in the lower part of Table 3.3. These control the assignment of user to user-groups, as well as removal of a user from a user-group. The meaning of $(ar, expr, \{g_1, g_2, g_3\}) \in canAssign$ is that member of administrator role ar (or senior) can assign any user-group in $\{g_1, g_2, g_3\}$ to a user which satisfy the conditions in expr. EXPR(UA $\cup$ UG) now includes the current membership or non-membership of user in user-groups along with user attributes. The policy language has the following changes.

$$set ::= att_{u_i}(u) \mid effective_{att_{u_i}}(u) \mid directUg(u) \mid effectiveUg(u) \mid constantSet$$

Table 3.7: Example Administrative Rules in canRemove UGA

Admin Role	Prereq. Cond	AllowedGroups
UniAdmin	graduated $\in \text{effective}_{\text{studStatus}}(\text{u}) \wedge$ {G,UGR} $\cap$ effectiveUg(u) $\neq$	{G,UGR}
DeptAdmin	COS $\notin \text{effective}_{\text{college}}(\text{u})$	{CSD}

$\text{atomic} ::= \text{constantAtomic}$

where effectiveUg(u) = directUg(u) $\cup$ ($\bigcup_{\forall \text{ug}_\text{i} \in \text{directUg(u)}} \{\text{ug}_\text{j} | \text{ug}_\text{i} \succeq_{ug} \text{ug}_\text{j}\}$)

The canRemove relation in Table 3.3 controls the removal of a user from user-group memberships. The remove operation is said to be weak in that it will only impact explicit memberships of user. A user is an explicit member of group ug if $ug \in$ directUg(u) whereas a user is an implicit member of ug if for some $ug_i \in$ directUg(u), $ug \in \{ug_j |\ ug_i \succeq_{ug}\ ug_j\}$ exists. It should be mentioned that removal of a user from any explicit membership ug will automatically result in removal from all implicit membership due to ug. Figure 3.5 shows assignment of user to user-group G allowed by first rule in Table 3.6. This assignment results in updates on effective attributes of user as user now inherits all attributes from group G along with direct attributes assigned through UAA. In case of weak removal (using Figure 3.1), suppose a user is an explicit member of groups CSD and G and administrator role DeptAdmin removes user from CSD (authorized by second rule in Table 3.7), the user will still have attributes of CSD through its membership in G.

3.3.4 Operational Specification

Table 3.8 outlines administrative operations required for user-group membership and attribute assignment. In all operations: $ar \in \text{AR}, u \in \text{U}, att_u \in \text{UA}, ug \in$ UG. A request (first column) succeeds only if a tuple exists in administrative relation and the entity satisfies the conditions (second column), in which case the update (third column) is performed.

3.3.5 GURA$_\text{G}$ Model Extensions

This section proposes some enhancements to GURA$_\text{G}$.

Table 3.8: Operational Specification

Operations	Conditions	Updates
In following operations: $VAL' \in 2^{\text{Range}(att_u)}$, $val \in VAL'$, $expr \in$ EXPR(UA)		
Add(ar, u, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in \mathbf{canAdd_{att_u}}$ $\wedge\ expr(u)$ = True $\wedge\ val \notin att_u(u)$	$att'_u(u)$ = $att_u(u) \cup \{val\}$
Delete(ar, u, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in \mathbf{canDelete_{att_u}}$ $\wedge\ expr(u)$ = True $\wedge\ val \in att_u(u)$	$att'_u(u)$ = $att_u(u) \setminus \{val\}$
Add(ar, ug, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in \mathbf{canAdd_{att_u}}$ $\wedge\ expr(ug)$ = True $\wedge\ val \notin att_u(ug)$	$att'_u(ug)$ = $att_u(ug) \cup \{val\}$
Delete(ar, ug, att_u, val)	if $\exists\langle ar, expr, VAL'\rangle \in \mathbf{canDelete_{att_u}}$ $\wedge\ expr(ug)$ = True $\wedge\ val \in att_u(ug)$	$att'_u(ug)$ = $att_u(ug) \setminus \{val\}$
In following operations: $UG' \in 2^{\text{UG}}$, $ug \in UG'$, $expr \in \text{EXPR}(\text{UA} \cup \text{UG})$		
Assign(ar, u, ug)	if $\exists\langle ar, expr, UG'\rangle \in \mathbf{canAssign}$ $\wedge\ expr(u)$ = True $\wedge\ ug \notin \text{directUg}(u)$	$\text{directUg}'(u)$ = $\text{directUg}(u) \cup \{ug\}$
Remove(ar, u, ug)	if $\exists\langle ar, expr, UG'\rangle \in \mathbf{canRemove}$ $\wedge\ expr(u)$ = True $\wedge\ ug \in \text{directUg}(u)$	$\text{directUg}'(u)$ = $\text{directUg}(u) \setminus \{ug\}$

Strong Removal: We can define a strong removal operation as per the following example using Figure 3.1. If a user is explicit member of CSD and G and administrator role DeptAdmin removes this user from CSD (allowed by second rule in Table 3.7), the user will also be removed from group G along with CSD if allowed by authorization rules. If the user cannot be deleted from G, the operation will have no effect.

Inherited Value Deletion in User: Let Alice have administrator role r_1 and Alice tries to delete inherited value val from attribute att_u of user u_1. Let there be a $canDelete_{att_u}$ rule $(r, cond, allowedVal)$ and if $r_1 \geqslant r, val \in allowedVal$ and u_1 satisfies $cond$, find all user groups ug in directUg(u_1) from where the attribute value val is inherited. There are two possibilities:(i) If there exists a $canRemove$ rule $(r, cond, allowedGroup)$ and if $r_1 \geqslant r, ug \in allowedGroup$ and u_1 satisfies the $cond$, remove u_1 from all such ug groups. (ii) If such a rule doesn't exist or u_1 cannot be removed from some ug groups, the operation will have no effect.

Inherited Value Deletion in User Group: Let Alice have role r_1, and Alice tries to delete inherited value val from attribute att_u of user group ug_1. Let there exists a $canDelete_{att_u}$ rule $(r, cond, allowedVal)$ and if $r_1 \geqslant r, val \in allowedVal$ and ug_1 satisfies $cond$, find all user groups ug junior to ug_1 which has val directly assigned. Delete val from all such ug as if Alice did this delete. If any delete fails this operation is aborted.

3.4 Group Based User Attribute Reachability Analysis

In ABAC, the attributes of an entity are critical in determining its permissions. Therefore, it is an important question to compute the attribute values that an entity can acquire through the combination of administrative roles and rules. In the context of $\mathrm{GURA_G}$, it is imperative to understand the set of attribute values a user can get based on direct assignment or via group memberships. Group hierarchy also exists in the HGABAC operational model which further complicates computation of the possible effective attribute values of a user. Although security administrators are trusted to assign attributes correctly, it is still desirable to understand the eventual set of attribute values that a user can acquire through multiple direct and indirect assignments. Such analysis can also help to identify a sequence of administrative actions required by administrators to assign certain attribute values to the users. It further allows administrators to know the future attribute values an entity can achieve based on predefined administrative rules, which can help them to understand if certain permissions can ever be granted to an entity.

In this section we analyze the attribute reachability analysis focusing on the effective attributes of the user achieved through direct assignment and through user-group memberships. This work extends the reachability analysis [107] done for GURA administrative model [106], where the attributes were only directly assigned to users without the concept of group memberships. In our analysis, we have defined a restricted $\mathrm{GURA_G}$ model, called $\mathrm{rGURA_G}$, which considers a subset of preconditions which can be created in $\mathrm{GURA_G}$. We abstract $\mathrm{rGURA_G}$ into a state transition system and specify three separate instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$—to cover different set of prerequisite conditions for attributes assignments to a user or a group, and also for

user to group membership assignment. Our reachability analysis primarily focuses on the effective set of attributes of users which is the union of direct attributes and attributes attained by group membership. We have defined reachability queries which is the required set of effective attributes a user can achieve in any target state. Two different types of reachability queries are discussed, one with the exact values and another with the superset of attribute values. We will show that the general reachability problem for $\mathrm{rGURA_G}$ schemes is PSPACE-complete. We further identify certain more restricted cases of $\mathrm{rGURA_G}$ schemes where the reachability problem can be solved in polynomial time. For such instances we will provide algorithms and a sequence of administrative requests (referred as reachability plan) to satisfy the reachability query.

3.4.1 $\mathrm{GURA_G}$ Model and Scheme

The $\mathrm{GURA_G}$ administrative model [88] was proposed to regulate the assignment of user attribute values in HGABAC model via direct user attributes, user-group attributes and user to group memberships. The $\mathrm{GURA_G}$ model has three sub models: user attribute assignment (UAA), user group attribute assignment (UGAA) and user to group assignment (UGA), which regulates the direct and effective attributes of users. It should be noted that user group hierarchy (UGH) is considered fixed in the system and is not modified. Each of these sub models have different sets of administrative relations and preconditions definition using policy language as discussed in following subsections. The main difference between GURA and $\mathrm{GURA_G}$ is that $\mathrm{GURA_G}$ includes the assignment of attributes to groups and user to group memberships. Further, the prerequisite conditions specified in $\mathrm{GURA_G}$ are more expressive, as it also checks the current effective attributes or effective group memberships of entities to make future assignments.

Administrative Requests

Definition 1 (**Administrative Requests**)**.** The attributes and group memberships of entities are changed by administrative request made by administrators with certain administrative roles as defined in Table 3.9, where AR is the finite set of administrative roles. The administrative request

Table 3.9: Administrative Requests

In the following requests: $ar \in \mathrm{AR},\ att \in \mathrm{UA},\ val \in \mathrm{SCOPE}_{att},\ u \in \mathrm{U},\ ug \in \mathrm{UG}$	
– For User Attributes	
	$\mathsf{add}(ar,\ u,\ att,\ val)$
	$\mathsf{delete}(ar,\ u,\ att,\ val)$
– For User Group Attributes	
	$\mathsf{add}(ar,\ ug,\ att,\ val)$
	$\mathsf{delete}(ar,\ ug,\ att,\ val)$
– For User to User-Group Membership	
	$\mathsf{assign}(ar,\ u,\ ug)$
	$\mathsf{remove}(ar,\ u,\ ug)$

$\mathsf{add}(ar,\ u,\ att,\ val)$ is made by administrator with role ar to add value val to attribute att of user u. Similar administrative request are used for groups also. Administrative requests assign and remove are required for managing group memberships. Each administrative request can add or delete a single attribute value from a user or group.

Definition 2 (**Administrative Rules).** Administrative rules are tuples in administrative relations which specify conditions under which administrative requests are authorized. All three sub-models (UAA, UGAA, UGA) in $\mathrm{GURA_G}$ model have administrative relations to define these rules.

The UAA sub-model deals with addition or deletion of attributes from the user. It has two administrative relations shown in Table 3.10, where a rule $\langle ar,\ c,\ val\rangle \in \mathsf{canAddU}_{att}$ authorizes request $\mathsf{add}(ar,\ u,\ att,\ val)$ if user u satisfies precondition c. Similarly, rule $\langle ar,\ c,\ val\rangle \in \mathsf{canDeleteU}_{att}$ authorizes $\mathsf{delete}(ar,\ u,\ att,\ val)$ requests if user u satisfies precondition c. In UAA, the precondition $c \in \mathrm{C}$ includes only current direct and effective attributes of user u. Similar relations also exist for administering attributes of groups as discussed in sub-model UGAA. In UGAA, $c \in \mathrm{C}$ involves current direct or effective attributes of the group whose attributes are modified.

The UGA sub-model has two relations shown in lower part of Table 3.10. The rule $\langle ar,\ c,\ ug\rangle \in \mathsf{canAssign}$ authorizes user to group assignment request $\mathsf{assign}(ar, u, ug)$ if user u satisfies the precondition c. Similarly rule $\langle ar,\ c,\ ug\rangle \in \mathsf{canRemove}$ authorizes remove request $\mathsf{remove}(ar, u, ug)$

Table 3.10: Redefined $\mathrm{GURA_G}$ Administrative Model

– User Attribute Assignment (**UAA**):
For each att in UA,

$$\mathsf{canAddU}_{att} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{SCOPE}_{att}$$
$$\mathsf{canDeleteU}_{att} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{SCOPE}_{att}$$

– User Group Attribute Assignment (**UGAA**):
For each att in UA,

$$\mathsf{canAddUG}_{att} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{SCOPE}_{att}$$
$$\mathsf{canDeleteUG}_{att} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{SCOPE}_{att}$$

– User to User Group Assignment (**UGA**):

$$\mathsf{canAssign} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{UG}$$
$$\mathsf{canRemove} \subseteq \mathrm{AR} \times \mathrm{C} \times \mathrm{UG}$$

if user u satisfies precondition c. The precondition $c \in \mathrm{C}$ involves both current direct or effective attributes and groups of user u.

The expressive power of the $\mathrm{GURA_G}$ model is primarily determined by the richness of the policy language used to define the preconditions C in Table 3.10. The most general language for this purpose is defined in [88], similar to the most general language of [106] (but without atomic attributes).

Note: In the original $\mathrm{GURA_G}$ definition [88], the administrative relations of Table 3.10 are defined with $2^{\mathrm{SCOPE}_{att}}$ substituted for SCOPE_{att} and 2^{UG} substituted for UG. With the modification of Table 3.10 the administrative relations can grow linearly in the size of SCOPE_{att} and UG. This does not materially impact the complexity analysis of the reachability problem.

$\mathbf{GURA_G}$ scheme

For purpose of our reachability analysis, we express the $\mathrm{GURA_G}$ model according to the notations developed in [166], following the treatment in [107]. The $\mathrm{GURA_G}$ scheme is presented as a state transition system where each state consists of direct attribute assignments for each attribute of every user and group, and also each user to groups membership. A transition between states occurs when an authorized administrative request changes either direct user or group attribute, or changes user

to group membership. The general definition for $\mathrm{GURA_G}$ scheme is as follows.

Definition 3 ($\mathrm{GURA_G}$ **Scheme).** A $\mathrm{GURA_G}$ scheme is a state transition system $\langle \mathrm{U}, \mathrm{UA}, \mathrm{AR}, \mathrm{SCOPE}, \mathrm{UG}, \succeq_{ug}, \Psi, \Gamma, \delta \rangle$ where,

(i) U, UA, AR, UG, $\succeq_{ug}$ are as defined in Tables 3.1 and 3.9.

(ii) $\mathrm{SCOPE} = \langle \mathrm{SCOPE}_{att_1} \ldots \mathrm{SCOPE}_{att_n} \rangle$ where $att_i \in \mathrm{UA}$, is the collection of scopes of all attributes.

(iii) Ψ is the collection of all administrative rules in UAA, UGAA and UGA sub-models.

(iv) Γ and δ are set of states and transition function respectively, defined in following parts of this subsection.

Direct State

Γ is the finite set of states where each state $\gamma \in \Gamma$ records directly assigned attributes of each user and user group, along with user to groups membership. The direct user attribute assignment in state γ, denoted by UAA_γ, contains tuples of the form $\langle u, att, val \rangle$ for every $u \in \mathrm{U}$ and every $att \in \mathrm{UA}$ such that $att(u) = val$ and $val \in \mathrm{Range}(att)$ in state γ. To ensure uniqueness of user attribute values we require the following.

$$\langle u, att, val_1 \rangle \in \mathrm{UAA}_\gamma \wedge \langle u, att, val_2 \rangle \in \mathrm{UAA}_\gamma \Rightarrow val_1 = val_2$$

Similarly, direct user group attribute assignment in state γ, denoted by UGAA_γ, contains tuples of the form $\langle ug, att, val \rangle$ for every $ug \in \mathrm{UG}$ and every $att \in \mathrm{UA}$ such that $att(ug) = val$ and $val \in \mathrm{Range}(att)$ in state γ, with the following uniqueness requirement.

$$\langle ug, att, val_1 \rangle \in \mathrm{UGAA}_\gamma \wedge \langle ug, att, val_2 \rangle \in \mathrm{UGAA}_\gamma \Rightarrow val_1 = val_2$$

Finally, direct user to group assignment in state γ, denoted UGA_γ, contains tuples of the form $\langle u, val \rangle$ for every $u \in \mathrm{U}$ such that $\mathrm{directUg}(u) = val$ and $val \in 2^{\mathrm{UG}}$ in state γ, with the following

uniqueness requirement.

$$\langle u, val_1 \rangle \in \mathrm{UGA}_\gamma \wedge \langle ug, val_2 \rangle \in \mathrm{UGA}_\gamma \Rightarrow val_1 = val_2$$

Note that information in a state can be used to calculate the effective attributes for user or group and effective user to groups membership in that state. For convenience we understand the notation $att_\gamma(u)$, $att_\gamma(ug)$ and $\mathrm{directUg}_\gamma(u)$ to denote the values of these functions in state γ for $u \in \mathrm{U}$ and $ug \in \mathrm{UG}$.

Transition Function

Any change in the direct state records $(\mathrm{UAA}_\gamma, \mathrm{UGAA}_\gamma, \mathrm{UGA}_\gamma)$ will transform the current state to a new state. The transition function specifies the change from one state to another in a $\mathrm{GURA_G}$ system based on current direct or effective values and administrative requests, as shown in Table 3.11. Formally, $\delta : \Gamma \times \mathrm{REQ} \rightarrow \Gamma$, where REQ is the set of possible administrative requests.

3.4.2 Restricted $\mathrm{GURA_G}$ ($\mathrm{rGURA_G}$)

In this section, we introduce a restricted form of $\mathrm{GURA_G}$ administrative model, called $\mathrm{rGURA_G}$, used in our attribute reachability analysis. This restricted form allows a subset of the precondition language defined for $\mathrm{GURA_G}$ [88], whereby our analysis also establishes lower bound results on the complexity analysis for richer $\mathrm{GURA_G}$ model. We first present a generalized policy language for $\mathrm{rGURA_G}$, followed by three specific instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$, and $\mathrm{rGURA_{G_{1+}}}$.

The left side of Figure 3.6 shows the relation between these schemes, while the right side shows the rGURA schemes discussed in [107]. At a high level, $\mathrm{rGURA_{G_0}}$ and $\mathrm{rGURA_{G_1}}$ add group attributes respectively to $\mathrm{rGURA_0}$and $\mathrm{rGURA_1}$, while $\mathrm{rGURA_{G_{1+}}}$ further adds administration of user membership in groups. Thereby, in $\mathrm{rGURA_{G_0}}$ and $\mathrm{rGURA_{G_1}}$ the administrative relations canAssign and canRemove are empty whereas they are populated in $\mathrm{rGURA_{G_{1+}}}$. Table 3.12 provides example administrative rules for each $\mathrm{rGURA_G}$ instance, as will be explained below.

Table 3.11: Transition Function

(1) γ_1 and γ_2 are the source and target states respectively.
(2) Let : $ar \in \text{AR},\ u \in \text{U},\ ug \in \text{UG},\ att \in \text{UA},\ val' \in \text{SCOPE}_{att},\ ug' \in \text{UG}$.
(3) Satisfy_u: $\text{U} \times \text{C} \times \Gamma \rightarrow \{\textbf{true}, \textbf{false}\}$, returns **true** if user $u \in \text{U}$ satisfies precondition $c \in \text{C}$ in state $\gamma \in \Gamma$, else **false**.
(4) $\text{Satisfy}_{ug} : \text{UG} \times \text{C} \times \Gamma \rightarrow \{\textbf{true}, \textbf{false}\}$, returns **true** if user group $ug \in \text{UG}$ satisfies precondition $c \in \text{C}$ in state $\gamma \in \Gamma$, else **false**.
(5) $\text{Satisfy}_{u\text{–}ug} : \text{U} \times \text{C} \times \Gamma \rightarrow \{\textbf{true}, \textbf{false}\}$, returns **true** if user $u \in \text{U}$ satisfies precondition $c \in \text{C}$ in state $\gamma \in \Gamma$, else **false**.

Request	Pre-Conditions	Target State
add(ar, u, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in$ canAddU$_{att}$. ($\text{Satisfy}_u(u,\ c,\ \gamma_1)$ $\wedge\ val' \notin att_{\gamma_1}(u)$)	$att_{\gamma_2}(u) = att_{\gamma_1}(u) \cup \{val'\}$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u)$, $\text{UAA}_{\gamma_2} = \text{UAA}_{\gamma_1} \setminus \langle u, att, att_{\gamma_1}(u) \rangle \cup \langle u, att, att_{\gamma_2}(u) \rangle$.
delete(ar, u, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in$ canDeleteU$_{att}$. ($\text{Satisfy}_u(u,\ c,\ \gamma_1)$ $\wedge\ val' \in att_{\gamma_1}(u)$)	$att_{\gamma_2}(u) = att_{\gamma_1}(u) \setminus \{val'\}$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u)$, $\text{UAA}_{\gamma_2} = \text{UAA}_{\gamma_1} \setminus \langle u, att, att_{\gamma_1}(u) \rangle \cup \langle u, att, att_{\gamma_2}(u) \rangle$.
add(ar, ug, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in$ canAddUG$_{att}$. ($\text{Satisfy}_{ug}(ug,\ c,\ \gamma_1)$ $\wedge\ val' \notin att_{\gamma_1}(ug)$)	$att_{\gamma_2}(ug) = att_{\gamma_1}(ug) \cup \{val'\}$, $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u)$, $\text{UGAA}_{\gamma_2} = \text{UGAA}_{\gamma_1} \setminus \langle ug, att, att_{\gamma_1}(ug) \rangle \cup \langle ug, att, att_{\gamma_2}(ug) \rangle$.
delete(ar, ug, att, val')	$\exists \langle ar,\ c,\ val' \rangle \in$ canDeleteUG$_{att}$. ($\text{Satisfy}_{ug}(ug,\ c,\ \gamma_1)$ $\wedge\ val' \in att_{\gamma_1}(ug)$)	$att_{\gamma_2}(ug) = att_{\gamma_1}(ug) \setminus \{val'\}$, $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u)$, $\text{UGAA}_{\gamma_2} = \text{UGAA}_{\gamma_1} \setminus \langle ug, att, att_{\gamma_1}(ug) \rangle \cup \langle ug, att, att_{\gamma_2}(ug) \rangle$.
assign(ar, u, ug')	$\exists \langle ar,\ c,\ ug' \rangle \in$ canAssign. ($\text{Satisfy}_{u\text{–}ug}(u,\ c,\ \gamma_1)$ $\wedge\ ug' \notin \text{directUg}_{\gamma_1}(u)$)	$\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u) \cup \{ug'\}$ $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\text{UGA}_{\gamma_2} = \text{UGA}_{\gamma_1} \setminus \langle u, \text{directUg}_{\gamma_1}(u) \rangle \cup \langle u, \text{directUg}_{\gamma_2}(u) \rangle$.
remove(ar, u, ug')	$\exists \langle ar,\ c,\ ug' \rangle \in$ canRemove. ($\text{Satisfy}_{u\text{–}ug}(u,\ c,\ \gamma_1)$ $\wedge\ ug' \in \text{directUg}_{\gamma_1}(u)$)	$\text{directUg}_{\gamma_2}(u) = \text{directUg}_{\gamma_1}(u) \setminus \{ug'\}$ $att_{\gamma_2}(u) = att_{\gamma_1}(u)$, $att_{\gamma_2}(ug) = att_{\gamma_1}(ug)$, $\text{UGA}_{\gamma_2} = \text{UGA}_{\gamma_1} \setminus \langle u, \text{directUg}_{\gamma_1}(u) \rangle \cup \langle u, \text{directUg}_{\gamma_2}(u) \rangle$.

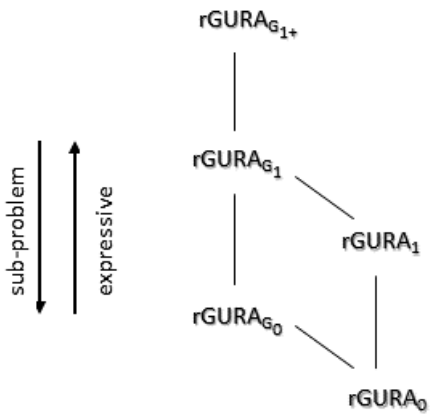


Figure 3.6: $\mathrm{rGURA_G}$ (Left Side) and rGURA (Right Side) Schemes

Definition 4 ($\mathrm{rGURA_G}$ **Scheme).** The $\mathrm{rGURA_G}$ scheme uses the policy grammar below, to specify preconditions C in Table 3.10,

$$\varphi ::= \neg\, \varphi \mid \varphi \wedge \varphi \mid svalue \in \mathrm{direct} \mid svalue \in \mathrm{effective}$$

$$svalue ::= \; sval_1 \mid sval_2 \mid \; \ldots \; \mid sval_m$$

where $\mathrm{SCOPE}_{att} = \{sval_1,\ sval_2,\ \ldots,\ sval_m\}$. The two non-terminals direct and $\mathrm{effective}$, are individually defined in its three instances—$\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$—in following subsections. Note that for convenience we denote effective attribute function for an attribute att as e_att in following discussions.

The $\mathrm{rGURA_{G_0}}$ Scheme

In $\mathrm{rGURA_{G_0}}$ scheme, preconditions for rules in $\mathsf{canAddU}_{att}$ and $\mathsf{canDeleteU}_{att}$ relations only allow the same attribute att whose value is added or deleted from user. Therefore, conditions for user u have non-terminals direct and $\mathrm{effective}$ defined as follows.

$$\mathrm{direct} ::= att(u) \quad \& \quad \mathrm{effective} ::= e\_att(u)$$

Similarly, the administrative relations in $\mathsf{canAddUG}_{att}$ and $\mathsf{canDeleteUG}_{att}$ for user group ug have direct and $\mathrm{effective}$ defined as follows.

$$\mathrm{direct} ::= att(ug) \quad \& \quad \mathrm{effective} ::= e\_att(ug)$$

Table 3.12: Example Rules in $\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_{1+}}}$ Schemes

Relation	Admin Role	Pre-requisite Condition	Value
Rules in $\mathbf{rGURA_{G_0}}$ scheme			
canAddU$_{skills}$	DeptAdmin	c $\in e\_skills(u)$ $\wedge \neg$ (java $\in$ *skills*(u))	c++
canDeleteU$_{roomAcc}$	BuildAdmin	3.02 $\in e\_roomAcc(u)$	1.2
canAddUG$_{college}$	UnivAdmin	$\neg$ (COE $\in$ *college*(ug))	COS
canDeleteUG$_{roomAcc}$	BuildAdmin	2.04 $\in e\_roomAcc(ug)$	2.03
Rules in $\mathbf{rGURA_{G_1}}$ scheme further add			
canAddU$_{studType}$	DeptAdmin	java $\in e\_skills(u)$ $\wedge$ 2.03 $\in$ *roomAcc*(u)	Grad
canDeleteU$_{roomAcc}$	BuildAdmin	3.02 $\in roomAcc(u)$ $\wedge$ COS $\in$ *college*(u)	3.02
canAddUG$_{skills}$	DeptAdmin	COS $\in$ *college*(ug) $\wedge$ UnderGrad $\in$ $e\_studType(ug)$	java
canDeleteUG$_{college}$	UnivAdmin	2.04 $\in e\_roomAcc(ug)$ $\wedge$ 2.03 $\in$ $e\_roomAcc(ug)$	BUS
Rules in $\mathbf{rGURA_{G_{1+}}}$ scheme further add			
canAssign	DeptAdmin	1.02 $\in e\_roomAcc(u)$ $\wedge \neg$ (BUS $\in$ *college*(u)) $\wedge$ $\mathrm{G_2} \in \mathrm{directUg}(u)$	$\mathrm{G_1}$
canRemove	GroupAdmin	$\mathrm{G_1} \in \mathrm{effUg}(u)$ $\wedge$ $\mathrm{G_2} \in \mathrm{directUg}(u)$	$\mathrm{G_2}$

The examples for $\mathrm{rGURA_{G_0}}$ shown in Table 3.12 conform to these restrictions. Note that the attribute being updated is given as the subscript in the Relation column and the conditions in the Pre-requisite Condition column only involve this attribute.

The $\mathbf{rGURA_{G_1}}$ Scheme

In $\mathrm{rGURA_{G_1}}$ scheme, the precondition can include any attribute from the set of attributes. Therefore, conditions in rules for canAddU$_{att}$ and canDeleteU$_{att}$ relations for user u have direct and effective defined as follows where $att_i \in \mathrm{UA}$.

direct ::= $att_i(u)$ & effective ::= $e\_att_i(u)$

Similarly, the conditions for user group ug in relations canAddUG$_{att}$ and canDeleteUG$_{att}$ have non-terminals direct and effective defined as follows.

direct ::= $att_i(ug)$ & effective ::= $e\_att_i(ug)$

The added rules for $\mathrm{rGURA_1}$in Table 3.12 illustrate this, where the preconditions involve attributes other than the one being updated. The earlier rules for $\mathrm{rGURA_{G_0}}$ continue to be valid for $\mathrm{rGURA_1}$.

The rGURA$_{G_{1+}}$ Scheme

The $\mathrm{rGURA}_{\mathrm{G}_{1+}}$ scheme allows changes in user group memberships besides modifying the attributes of user and user groups. Therefore, in addition to the grammar supported by $\mathrm{rGURA}_{\mathrm{G}_1}$ scheme, $\mathrm{rGURA}_{\mathrm{G}_{1+}}$ also includes user's direct or effective group memberships as preconditions in rules for canAssign and canRemove administrative relations. The additional grammar to specify such preconditions is specified below:

$$\varphi ::= ug \in \mathrm{directUg}(u) \mid ug \in \mathrm{effUg}(u)$$

In Table 3.12, rule in canAssign includes effective values for $roomAcc$, direct values for $college$ attribute and direct groups of user u.

3.4.3 Reachability Problem Definition

In this section, we provide a formal definition of our attribute reachability problem along with the reachability query and different query types supported in our analysis. The general approach is similar to that of [107], except that atomic-valued attributes are excluded (as noted in subsection 3.2.2) and reachability is defined with respect to effective rather than direct attributes ([107] does not have the notion of effective attributes).

The user attribute reachability analysis problem (or reachability problem) is based on the effective attributes of the user. Informally, the problem can be stated as: Given an initial transition system state with a set of attribute assignments of the user, the user's group memberships and the attributes of all the user's member groups, can administrators with a given set of administrative roles issue one or more administrative requests, which transition to a target state having the set of specified effective attributes for that user? We highlight some simplifications in our reachability analysis process. First, as the changes made to the attributes or group memberships of one user do not affect the attributes or group memberships of another user, our analysis will only determine the effective attributes of a single user of interest and hence will only consider attribute assignment of that user, its group memberships and attributes of these groups. Formally, we assume U = {u}

in our analysis [107]. Second, as the reachability analysis focuses on powers of a certain set of administrative roles $\mathrm{SUBAR} \subseteq \mathrm{AR}$, we do not consider the administrative rules specified for roles outside of SUBAR. In other words, we can assume $\mathrm{AR} = \mathrm{SUBAR}$. These simplifications gives our analysis more convenient statements without loosing generality.

Definition 5 (**Reachability Query**)**.** A reachability query $q \in \mathrm{Q}$ specifies a subset of effective values of a user for some attributes in any target state. Formally,

$$q \subseteq \{\langle u, e\_att, vset\rangle \mid u \in \mathrm{U}, att \in \mathrm{UA}, vset \in \mathrm{Range}(att)\}$$

In the example problems discussed later in subsection 3.5.3, we will use the following notation to specify our query, which is equivalent to the notation defined above.

$$q \subseteq \{e\_att(u) = vset \mid u \in \mathrm{U}, att \in \mathrm{UA}, vset \in \mathrm{Range}(att)\}$$

For example,

$q = \{\langle \mathrm{u}, \mathrm{e\_roomAcc}, \{2.04\}\rangle, \langle \mathrm{u}, \mathrm{e\_skills}, \{\mathrm{c}\}\rangle, \langle \mathrm{u}, \mathrm{e\_college}, \{\mathrm{COS}, \mathrm{COE}\}\rangle\}$ is equivalent to

$q = \{\mathrm{e\_roomAcc(u)} = \{2.04\}, \mathrm{e\_skills(u)} = \{\mathrm{c}\}, \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}\}\}$.

Two types of reachability query are defined in the system. A query is called "strict" satisfied if every effective attribute value specified in the query is exactly the same as that in the target state. A query is called "relaxed" satisfied by the user if in the target state every effective attribute value of the user is a superset of the corresponding attribute values specified in the reachability query. For example, let UA = {skills}, U = {Bob} and reachability query q = ⟨Bob, e_skills, {c, java}⟩. For strict query type, q can be satisfied in states $\gamma' \in \Gamma$ where $e\_skills_{\gamma'}(u)$ = {c, java}. In relaxed query type, q can be satisfied by any state $\gamma'' \in \Gamma$ where $e\_skills_{\gamma''}(u)$ = $setval$ and {c, java} ⊆ $setval$. For ease of understanding, we represent the effective value of attribute att for user u in state $\gamma \in \Gamma$ as $e\_att_{\gamma}(u)$. The formal definition for reachability query types is given below.

Definition 6 (**Reachability Query Types**)**.** For any $\mathrm{rGURA_G}$ scheme $\langle \mathrm{U}, \mathrm{UA}, \mathrm{AR}, \mathrm{SCOPE}, \mathrm{UG}, \succeq_{ug}, \Psi, \Gamma, \delta\rangle$, we formally define two Reachability Query Types as:

- $\mathrm{RP}_{=}$ or strict satisfied queries have the entailment function $\vdash_{\mathrm{RP}_{=}}: \Gamma \times \mathrm{Q} \rightarrow \{\mathbf{true}, \mathbf{false}\}$ which returns $\mathbf{true}$ (i.e., $\gamma \vdash_{\mathrm{RP}_{=}} q$) if $\forall\ \langle u, e\_att, vset\rangle \in q.\ e\_att_{\gamma}(u) = vset$.

- $\mathrm{RP}_{\supseteq}$ or relaxed satisfied queries have the entailment function $\vdash_{\mathrm{RP}_{\supseteq}}: \Gamma \times \mathrm{Q} \rightarrow \{\mathbf{true}, \mathbf{false}\}$ which returns $\mathbf{true}$ (i.e., $\gamma \vdash_{\mathrm{RP}_{\supseteq}} q$) if $\forall\ \langle u, e\_att, vset\rangle \in q.\ e\_att_{\gamma}(u) \supseteq vset$.

It is clear that given a scheme and problem instance, if $\mathrm{RP}_{=}$ query problem is satisfied then $\mathrm{RP}_{\supseteq}$ problem is also satisfied, but not vice versa. The following two definitions are same as defined in [107], but we will state them for the sake of completeness.

Definition 7 (**Reachability Plan**)**.** A Reachability Plan or plan is a sequence of authorized administrative requests to transition from initial state to the target state. For any $\mathrm{rGURA_G}$ scheme $\langle \mathrm{U}, \mathrm{UA}, \mathrm{AR}, \mathrm{SCOPE}, \mathrm{UG}, \succeq_{ug}, \Psi, \Gamma, \delta\rangle$ and states $\gamma_0, \gamma' \in \Gamma$, reachability plan is a sequence of authorized requests $\langle\ req_1, req_2, \ldots, req_n\rangle$ where $req_i \in \mathrm{REQ}$ $(1 \leq i \leq n)$, to transition from an initial state γ_0 to target state γ' if: $\gamma_0 \overset{req_1}{\rightarrow} \gamma_1 \overset{req_2}{\rightarrow} \gamma_2 \ldots \overset{req_n}{\rightarrow} \gamma'$. The arrow denotes a successful transition from one state to another due to an administrative request req_i authorized by rules in Ψ. We write $\gamma_0 \overset{plan_{\Psi}}{\rightsquigarrow} \gamma'$ to abbreviate the complete plan.

Informally, a reachability problem deals if there exists a reachability plan to transition from an initial state to some target state where the effective attribute values of the user satisfy a particular reachability query. Formally,

Definition 8 (**Reachability Problems**)**.** Given any $\mathrm{rGURA_G}$ scheme $\langle \mathrm{U}, \mathrm{UA}, \mathrm{AR}, \mathrm{SCOPE}, \mathrm{UG}, \succeq_{ug}, \Psi, \Gamma, \delta\rangle$, the attribute reachability problem is as follows:

- $\mathrm{RP}_{=}$ or strict reachability problem instance I is of the form $\langle \gamma_0, q\rangle$ where $\gamma_0 \in \Gamma$, $q \in \mathrm{Q}$ and checks if there exist a reachability plan P such that $\gamma_0 \overset{P_{\Psi}}{\rightsquigarrow} \gamma'$ and $\gamma' \vdash_{\mathrm{RP}_{=}} q$.
- $\mathrm{RP}_{\supseteq}$ or relaxed reachability problem instance I is of the form $\langle \gamma_0, q\rangle$ where $\gamma_0 \in \Gamma$, $q \in \mathrm{Q}$ and checks if there exist a reachability plan P such that $\gamma_0 \overset{P_{\Psi}}{\rightsquigarrow} \gamma'$ and $\gamma' \vdash_{\mathrm{RP}_{\supseteq}} q$.

3.4.4 PSPACE-Complete Reachability

In this section, we present our reachability analysis results for different $\mathrm{rGURA_G}$ schemes shown in Figure 3.6. These results are extensions to the results from GURA reachability analysis [107]

and also considers groups for assigning attributes to its member users. Our analysis will prove that $\mathrm{rGURA_G}$ schemes in Figure 3.6 in general are PSPACE-complete. For such schemes we will first show that all $\mathrm{rGURA_G}$ schemes are in PSPACE and then reduce a known PSPACE-complete problem to our problem schemes. In the next section, we also will provide polynomial algorithms for some restricted $\mathrm{rGURA_G}$ problem classes.

Lemma 1. *Reachability problem for every* $\mathrm{rGURA_G}$ *scheme in Figure 3.6 is in PSPACE.*

Proof. Each state of a non-deterministic Turing machine stores some information to predict future states. This information takes polynomial amount of space and therefore all instance are in PSPACE. This proof is similarly stated for GURA schemes in [107], however we will discuss it for the sake of completeness.

The proof is an extension to proof discussed in [107]. A Non-deterministic Turing machine can be used to implement following algorithm for each $\mathrm{rGURA_G}$ problem instance. Each state of the machine stores information to determine next possible states it can enter. In $\mathrm{rGURA_G}$ schemes, this information consists of current direct user attribute assignments, direct group attribute assignments, user to group assignments, attribute scopes, administrative rules, user groups and reachability query. The administrative rules are applied against current user or group attributes, or user to group assignments to get all next possible states. In each future state, Turing machine checks against the reachability query, and determines if the query is satisfied. If in a state the query is satisfied, Turning machine comes to halt or otherwise, same process repeats till a satisfied state is reached or it is concluded that the query is non-satisfiable. The size of each state is bounded by input to the state. It is understandable that polynomial amount of space is required to store information required in each state of Non-deterministic Turing machine. Hence, each $\mathrm{rGURA_G}$ problem scheme in Figure 3.6 is in NPSPACE and therefore in PSPACE using Savitch's theorem [149]. □

Since all $\mathrm{rGURA_G}$ schemes are in PSPACE, it will now be sufficient to prove that all $\mathrm{rGURA_G}$ schemes are PSPACE-hard, which will conclude that the schemes are PSPACE-complete.

Corollary 1. *Reachability query types* $\mathrm{RP}_{\supseteq}$ *and* $\mathrm{RP}_{=}$ *for* $\mathrm{rGURA}_{\mathrm{G}}$ *schemes in general is PSPACE-complete.*

Proof. Recall that Figure 3.6 defines the relation between different $\mathrm{rGURA}_{\mathrm{G}}$ schemes and rGURA_0. The reachability analysis for rGURA_0 scheme discussed in [107] describes the scheme is PSPACE-complete. This scheme only allows change in attributes of the user. With respect to $\mathrm{rGURA}_{\mathrm{G}_0}$, it can be said that rGURA_0 scheme is a sub-problem without user groups. Therefore, the reduction from known PSPACE-complete problem (rGURA_0) to $\mathrm{rGURA}_{\mathrm{G}_0}$ is straightforward, which makes $\mathrm{rGURA}_{\mathrm{G}_0}$ as PSPACE-hard. Further, using Lemma 1, it is justified to claim that $\mathrm{rGURA}_{\mathrm{G}_0}$ is in PSPACE-complete.

Similar claim can also be made for $\mathrm{rGURA}_{\mathrm{G}_1}$ scheme where $\mathrm{rGURA}_{\mathrm{G}_0}$ is its sub-problem involving only the same attribute in preconditions for rules (Ψ). Therefore, $\mathrm{rGURA}_{\mathrm{G}_1}$ is PSPACE-hard and using Lemma 1, it is also PSPACE-complete. The analysis for $\mathrm{rGURA}_{\mathrm{G}_{1+}}$ is also alike the above two schemes where $\mathrm{rGURA}_{\mathrm{G}_1}$ is a sub-problem of $\mathrm{rGURA}_{\mathrm{G}_{1+}}$, therefore, $\mathrm{rGURA}_{\mathrm{G}_{1+}}$ is in PSPACE-hard and hence PSPACE-complete also. □

To explain further, we provide a brief overview of the analysis done in [107] for rGURA_0 scheme. The analysis result for $\mathrm{RP}_{\supseteq}$ query in rGURA_0 scheme is derived by reducing from role reachability problem. This role reachability problem for miniARBAC97 is proven PSPACE-complete [148], which by reduction makes rGURA_0 scheme as PSPACE-hard and using the same Lemma as 1, rGURA_0 scheme is PSPACE-complete. This reduction uses role as one of the many attributes and map the administrative rules of miniARBAC97 to the corresponding rules in rGURA_0 as their expressive power is same. Our $\mathrm{rGURA}_{\mathrm{G}_0}$ scheme extends rGURA_0 by introducing the notion of user groups and there corresponding administrative rules. Since, without user groups rGURA_0 and $\mathrm{rGURA}_{\mathrm{G}_0}$ are same, the results for $\mathrm{RP}_{\supseteq}$ in rGURA_0 still hold true and provide lower bound analysis. Therefore, we conclude that $\mathrm{RP}_{\supseteq}$ for [$\mathrm{rGURA}_{\mathrm{G}_0}$] is PSPACE-hard and using Lemma 1, it is in PSPACE-complete.

In GURA reachability [107], $\mathrm{RP}_{=}$ results for rGURA_0 uses reduction from SAS planning problem [43] in artificial intelligence. Each state variable in [SAS, U, B] problem (proved to be

PSPACE-complete in [43]) is mapped to one value in scope of attribute att. The operators which update the state variables to true or false are mapped to administrative rules. This reduction is polynomial time which results $\mathrm{RP}_{=}$ for rGURA_0 in PSPACE-complete. Same results can be extended for $\mathrm{rGURA}_{\mathrm{G}_0}$, where the values in scope for att will be for the user and each user-group. The operators will be mapped to administrative rules for user and each user-group. Also each operator will change only one variable holding [SAS, U, B] restrictions and the reduction is in polynomial time. Henceforth, $\mathrm{rGURA}_{\mathrm{G}_0}$is PSPACE-hard problem and therefore PSPACE-complete using Lemma 1.

3.5 Polynomial Reachability for Restricted Cases

In previous subsection, we proved that attribute reachability for any $\mathrm{rGURA}_{\mathrm{G}}$ scheme in general is PSPACE-complete. However, we have identified some instances of $\mathrm{rGURA}_{\mathrm{G}}$ schemes which can be solved in polynomial time under precondition restrictions on administrative rules (Ψ). Similar to [107], the following extended restrictions with broader semantic meaning are considered where $\overline{\mathrm{D}}$ and SR_{d} are always imposed together:

- **No negation ($\overline{\mathrm{N}}$)**: Ψ satisfies $\overline{\mathrm{N}}$ if no administrative rules in Ψ use negation in preconditions.
- **No deletion ($\overline{\mathrm{D}}$)**: Ψ satisfies $\overline{\mathrm{D}}$ if for each attribute $att \in \mathrm{UA}$, $\mathsf{canDeleteU}_{att}$ and $\mathsf{canDeleteUG}_{att}$ are empty. Further, canRemove rules are also empty, meaning, attribute values or groups once added cannot be deleted.
- **Single rule with direct values (SR_{d})**: Ψ satisfies SR_{d} if for each attribute $att \in \mathrm{UA}$, there is at most one precondition associated with a particular value assignment in rules of $\mathsf{canAddU}_{\mathsf{att}}$ or $\mathsf{canAddUG}_{\mathsf{att}}$. Therefore, an attribute value pair can either be added through user directly or through groups but not both. Similar, condition also exists for canAssign rules. Further, only direct conjuncts i.e. $val \in att_i(u)$, $val \in att_i(ug)$ or $ug \in \mathrm{directUg}(u)$ are allowed in prerequisite condition.

These restrictions are important in different kinds of attributes and scenarios. For instance,

No negation ($\overline{\mathrm{N}}$) restrictions have significance when attributes like $course$ or $degree$ are added to entities. It is likely that adding a new value for $course$ attribute do not require negation of another course as the precondition. Similarly, **No deletion** ($\overline{\mathrm{D}}$) restriction can apply for attribute like $skills$ where a value once added to any entity will never be deleted. The $\mathrm{SR_d}$ restriction allows only unique preconditions in administrative relations for user and user groups. This restriction essentially separates set of attributes into two parts, one which can be assigned only to user directly and others assigned through groups. For example, attribute like $roomAccess$ can be assigned through group as it is usually common to all users with certain characteristics, and if value changes for one user, it will change for all others too. Attribute like $advisor$ is assigned individually to each user as change for one user may not change it in others. Therefore, such restrictions are relevant in real world applications.

We now discuss reachability analysis for restricted $\mathrm{rGURA_G}$ schemes. The notation [$\mathrm{rGURA_{G}}x$, `Restriction`] specifies special instances of $\mathrm{rGURA_G}$ scheme where subscript x takes a value in 0, 1 or 1+ representing – $\mathrm{rGURA_{G_0}}$, $\mathrm{rGURA_{G_1}}$ or $\mathrm{rGURA_{G_{1+}}}$ and `Restriction` represents combinations of $\overline{\mathrm{N}}$, $\overline{\mathrm{D}}$ and $\mathrm{SR_d}$ specifying that administrative rules Ψ in the scheme satisfy these restrictions. For example, [$\mathrm{rGURA_{G_0}}-\overline{\mathrm{N}}$] denotes $\mathrm{rGURA_{G_0}}$ scheme where rules in Ψ satisfy $\overline{\mathrm{N}}$.

As shown in Figure 3.6, $\mathrm{rGURA_{G_{1+}}}$ scheme is the most expressive scheme where new attribute values are achieved by direct assignment to the user or to its effective groups, and also by changing user to group memberships. It is clear from the previous discussions that the scheme covers $\mathrm{rGURA_{G_1}}$ and $\mathrm{rGURA_{G_0}}$, which only allow change in attributes of the user or its effective groups. Therefore, we will only discuss algorithm for restricted $\mathrm{rGURA_{G_{1+}}}$ scheme which can be easily used for other two schemes by simply ignoring irrelevant administrative rules.

3.5.1 Reachability plan for $\mathrm{RP_=}$ in [$\mathrm{rGURA_{G_{1+}}}-\overline{\mathrm{N}}$]

First we will discuss reachability query type $\mathrm{RP_=}$ for $[\mathrm{rGURA_{G_{1+}}}-\overline{\mathrm{N}}]$ scheme which can be solved in polynomial time by Algorithm 1. This algorithm extends the algorithm discussed for

Algorithm 1 Plan Generation for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}$]

Input: problem instance I = $\langle \gamma_0, q \rangle$ **Output:** $plan$ or false
$plan$:= $\langle\rangle$; ▷ Initialize plan
$s := \gamma_0$; ▷ Initialize with state s
▷ Check if state s has more values than query
if ($\exists\ att \in \mathrm{UA}\ \exists\ \langle u, e\_att, vset \rangle \in q$). $e\_att(u) - vset \neq \emptyset$ **then return** false;
▷ Assign attribute values required in query to the user or its effective groups
while ($s \nvdash_{\mathrm{RP}_{=}} q \wedge$
  $((\exists\ att' := att \in \mathrm{UA}\ \exists\ rule := \langle ar, c, val \rangle \in \mathsf{canAddU}_{att'})$. ($\mathrm{Satisfy}_u(u,\ c,\ s) \wedge val \notin att'(u) \wedge$
    $\exists\ \langle u, e\_att', vset \rangle \in q.\ val \in vset$)) $\vee$
  $((\exists\ att' := att \in \mathrm{UA}\ \exists\ rule := \langle ar, c, val \rangle \in \mathsf{canAddUG}_{att'})$. ($\exists\ ug' := ug \in \mathrm{effUg}(u)$. $\mathrm{Satisfy}_{ug}(ug',\ c,\ s)$
    $\wedge\ val \notin att'(ug') \wedge \exists\ \langle u, e\_att', vset \rangle \in q.\ val \in vset$))
  $\vee$
  $((\exists\ ug'' := ug \in \mathrm{UG}\ \exists\ rule := \langle ar, c, ug'' \rangle \in \mathsf{canAssign})$. ($\mathrm{Satisfy}_{u-ug}(u,\ c,\ s) \wedge ug'' \notin \mathrm{directUg}(u) \wedge$
    $\forall\ att \in \mathrm{UA}\ \exists \langle u, e\_att, vset \rangle \in q.\ e\_att(ug'') \subseteq vset$))) **do**
    $s := s \ll rule$; ▷ apply rule on state s
  ▷ append administrative request to plan
  switch
    case $rule \in \mathsf{canAddU}_{att'}$:
      $plan := plan.\mathbf{append}(\mathsf{add}(ar, u, att', val))$;
    break;
    case $rule \in \mathsf{canAddUG}_{att'}$:
      $plan := plan.\mathbf{append}(\mathsf{add}(ar, ug', att', val))$;
    break;
    case $rule \in \mathsf{canAssign}$:
      $plan := plan.\mathbf{append}(\mathsf{assign}(ar, u, ug''))$;
    break;
end while
▷ check if reachability query is satisfied
if $s \vdash_{\mathrm{RP}_{=}} q$ **then return** $plan$ **else return false end if**

rGURA_1 [107] by including user group attribute assignments and also modification in user to group memberships. The added restriction to this scheme ($\overline{\mathrm{N}}$) requires preconditions in rules without negation conjuncts and therefore, administrative rules cannot specify addition of new attributes based on the absence of some other values. Hence, the current attribute values of user or groups are not required to be removed for adding new values or group, which precludes the need for investigating any $\mathsf{canDeleteU}_{att}$, $\mathsf{canDeleteUG}_{att}$ and canRemove rules.

The algorithm starts with the current set of attribute values and group memberships for user, and the attribute values for its member groups. It traverse all relevant $\mathsf{canAddU}_{att}$, $\mathsf{canAddUG}_{att}$ or canAssign rules to add new values to the attributes of user or to its effective user groups and also add new groups to the user. Since, the query type is restricted, the algorithm first checks if the current

effective attributes of user are not more than what required in the query (line 4). If the current values are extra, the algorithm returns false, since there are no delete administrative relations to delete such values. The while loop (line 5–24) terminates when either the query is satisfied or when no other values can be added from the rules in $\mathsf{canAddU}_{att}$ and $\mathsf{canAddUG}_{att}$ or no new groups can be added to the user using canAssign rules. When adding a new value to the user or its effective groups, the corresponding value must be checked against the query. If the value is present in the query, the addition is allowed. Similar check is also done to add new groups the user, where all the attributes present in the group should also be a part of the query. The order to add these values or new groups is independent to each other, since no negation conjuncts are required and presence of extra values in user or group will not stop from adding new values. Also, if later a new value is added to an entity, the while loop will again consider the relevant rules to add values based on the already added values. When a new attribute is added to user, its effective groups or a new group is assigned to the user, its corresponding administrative request is appended to the reachability plan $plan$. If the query is satisfied, the algorithm returns the corresponding reachability plan $plan$ or returns false stating that the query is unsatisfiable and user will not achieve desired effective attributes as mentioned in query.

Theorem 1. *Reachability query type* $\mathrm{RP}_{=}$ *for scheme* $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}]$ *is P.*

Proof. Algorithm 1 describes the polynomial time algorithm.

Complexity: The complexity is determined by the number of times the administrative rules in $\mathsf{canAddU}_{att}$, $\mathsf{canAddUG}_{att}$ or canAssign are traversed. If only one value is added by each of the rules, the complexity of Algorithm 1 is $\mathcal{O}(|\mathsf{canAssign}| \times |\mathrm{UG}| + ((\sum_{att \in \mathrm{UA}} |\mathrm{SCOPE}_{att}|) \times (\ |\mathsf{canAddU}_{\mathsf{att}}| + |\mathsf{canAddUG}_{\mathsf{att}}| \times |\mathrm{UG}|\ )))$, where $|\mathsf{canAddU}_{att}|$, $|\mathsf{canAddUG}_{att}|$ and $|\mathsf{canAssign}|$ represents number of the administrative rules in these relations and $|\mathrm{UG}|$ represents the maximum number of groups assigned to the user. Clearly, the complexity of algorithm is polynomial. □

The $\mathrm{RP}_{\geq}$ query type for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}]$ also has a polynomial algorithm, where the extra conditions to check the query before adding new values is removed since we can have values even

Algorithm 2 Group Assignment Plan Generation for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$

1: **Input:** problem instance I = $\langle \gamma_0, q \rangle$ **Output:** $plan_{ug}$
2: **if** $\gamma_0 \vdash_{\mathrm{RP}_{=}} q$ **then return** $plan_{ug}$:= $\langle\rangle$; ▷ Check initial state
3: $G_{ug} := \langle V_{ug}, E_{ug} \rangle$; V_{ug} := $\{ug \mid \exists\ ug \in \mathrm{UG}.\ \exists\ ug \notin \mathrm{directUg}(u).\ \exists\ \langle ar, c, ug \rangle \in \mathsf{canAssign}(u).\ \forall\ att \in \mathrm{UA}\ \exists \langle u, e\_att, vset \rangle \in q.\ e\_att(ug) \subseteq vset\ \}$; E_{ug} := $\emptyset$; ▷ Construct a directed graph
4: **for** each pair of nodes $((ug_1, ug_2) \in V_{ug})$ **do**
5: **if** $(\exists \langle ar, c, ug_2 \rangle \in \mathsf{canAssign}$. "$(ug_1 \in \mathrm{directUg}(u))$" is a conjunct in $c) \vee$
6: $(\exists \langle ar, c, ug_1 \rangle \in \mathsf{canAssign}$. "$\neg(ug_2 \in \mathrm{directUg}(u))$" is a conjunct in $c)$)
7: **then** E_{ug} := $E_{ug} \cup \{\langle ug_1, ug_2 \rangle\}$; **end if** ▷ Add edges
8: **end for**
9: **if** graph G_{ug} has cycles **then** remove the cyclic paths and $plan_{ug}$:= sequence of assign requests corresponding to the topological sort of G_{ug};

if they are not required in the query. The complexity will remain the same as shown in Theorem 1. Similar algorithm can also be devised for $\mathrm{RP}_{=}$ and $\mathrm{RP}_{\supseteq}$ query type in $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{N}}]$ and $[\mathrm{rGURA}_{\mathrm{G}_0} - \overline{\mathrm{N}}]$ schemes where canAssign rules will not be considered into the while loop for adding new groups to the user. Hence these schemes can be also solved in polynomial time.

3.5.2 Reachability plan for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$

We will now consider another restricted instance for $\mathrm{rGURA}_{\mathrm{G}_{1+}}$, $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ which can be solved by Algorithm 2 and 3. The scheme has two restrictions, $\overline{\mathrm{D}}$ which removes the need to consider delete administrative relations – $\mathsf{canDeleteU}_{att}$, $\mathsf{canDeleteUG}_{att}$ and canRemove. The SR_{d} restriction allows single preconditions for each attribute value pair or user group, with only direct values as conjuncts in preconditions. This restriction results in rules which can be either satisfied by user or any of its effective groups but not both. We have divided the algorithm into two algorithm for ease of understanding and to show how these algorithms can be reused in other schemes also.

Algorithm 2 is used to add new groups to the user. Since the preconditions only involves user's direct groups as conjuncts (SR_{d} restriction), the addition of groups is independent of the attributes and can be calculated separately. The administrative rules in this scheme can have negation conjuncts in preconditions, therefore, the order of assigning new groups can be mutually dependent. The algorithm first creates a directed graph where vertices V_{ug} are user groups and edges E_{ug} are directed based on conjuncts in precondition of rules in canAssign. In line 3, before adding a group

Algorithm 3 Plan Generation for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_1}$– $\overline{\mathrm{D}}$, SR_{d}]

Input: problem instance I = $\langle \gamma_0, q \rangle$ **Output:** $plan$ or false
if $\gamma_0 \vdash_{\mathrm{RP}_=} q$ **then return** $plan := \langle\rangle$; ▷ Check initial state
$toadd := \{(att, val) \mid att \in \mathrm{UA}, \langle u, e\_att, vset \rangle \in q, val \in vset\}$ ▷ Values required in query
$cur_u := \{(att, val) \mid att \in \mathrm{UA}, val \in att(u)\}$ ▷ Current values of user
▷ Find current values of user's effective groups
for each $ug \in \mathrm{effUg(u)}$ **do** $cur_{ug} := \{(att, val) \mid att \in \mathrm{UA}, val \in att(ug)\}$ **end for**
▷ Check if state γ_0 has more values than query
if $(cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug})) - toadd \neq \emptyset$ **then return** false;
$ppre_u := \emptyset$; **for** each $ug \in \mathrm{effUg}(u)$ **do** $ppre_{ug} := \emptyset$; **end for**
for (each $(att, val) \in toadd \cup ppre_u$) **do** ▷ Positive precondition values for user
  $ppre'_u := \{(att_1, val_1) \mid \exists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att}$. "$val_1 \in att_1(u)$" is a conjunct in $c\}$;
  $ppre_u := (\ ppre_u \cup (ppre'_u \setminus ppre_u\ )) \setminus cur_u$;
end for
for (each $ug \in \mathrm{effUg}(u)$) **do** ▷ Positive precondition values for effective groups
  for (each $(att, val) \in toadd \cup ppre_{ug}$) **do**
    $ppre'_{ug} := \{(att_1, val_1) \mid \exists \langle ar, c, val \rangle \in \mathsf{canAddUG}_{att}$. "$val_1 \in att_1(ug)$" is a conjunct in $c\}$;
    $ppre_{ug} := (\ ppre_{ug} \cup (ppre'_{ug} \setminus ppre_{ug})) \setminus cur_{ug}$;
  end for
end for
▷ Check if rules exists for values required
if $((\exists (att, val) \in toadd \cup ppre_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} ppre_{ug}) \setminus (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug}))).\ \nexists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att} \cup \mathsf{canAddUG}_{att})$**then return** false;
▷ Find negation values in rules required to add values for the user and its effective groups
$npre_u := \{(att_1, val_1) \mid \exists\ (att, val) \in (toadd \cup ppre_u) \setminus cur_u.\ \exists \langle ar, c, val \rangle \in \mathsf{canAddU}_{att}.$ "$\neg(val_1 \in att_1(u))$" is a conjunct in $c\}$
for each $ug \in \mathrm{effUg}(u)$ **do** $npre_{ug} := \{(att_1, val_1) \mid \exists\ (att, val) \in (toadd \cup ppre_{ug}) \setminus cur_{ug}\ .\ \exists \langle ar, c, val \rangle \in \mathsf{canAddUG}_{att}$. "$\neg(val_1 \in att_1(ug))$" is a conjunct in $c\}$ **end for**
▷ Negation in current values
if $((npre_u \cap cur_u \neq \emptyset) \vee (\forall\ ug \in \mathrm{effUg}(u).\ npre_{ug} \cap cur_{ug} \neq \emptyset))$ **then return** false;
▷ Construct a directed graph
$G := \langle V, E \rangle$; $V := toadd \cup ppre_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} ppre_{ug}) \setminus (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug}))$; $E := \emptyset$;
for each pair of nodes $((att_1, val_1), (att_2, val_2)) \in V$ **do**
  if $(((\exists \langle ar, c, val_2 \rangle \in \mathsf{canAddU}_{att_2}$. "$(val_1 \in att_1(u))$" is a conjunct in $c) \vee$
  $(\exists \langle ar, c, val_1 \rangle \in \mathsf{canAddU}_{att_1}$. "$\neg(val_2 \in att_2(u))$" is a conjunct in $c))$
  $\vee$
  $((\exists\ ug \in \mathrm{effUg}(u)).\ ((\exists \langle ar, c, val_2 \rangle \in \mathsf{canAddUG}_{att_2}$. "$(val_1 \in att_1(ug))$" is a conjunct in $c) \vee$
  $(\exists \langle ar, c, val_1 \rangle \in \mathsf{canAddUG}_{att_1}$. "$\neg(val_2 \in att_2(ug))$" is a conjunct in $c))))$
  then $E := E \cup \{\langle (att_1, val_1), (att_2, val_2) \rangle\}$; **end if** ▷ Add edges to the graph
end for
$valset := toadd - (cur_u \cup (\bigcup_{ug \in \mathrm{effUg}(u)} cur_{ug})$); ▷ Values in query not in state γ
if $\exists (att_1, val_1) \in valset\ \exists \langle (att, val), (att_1, val_1) \rangle \in E.\ (att, val) \notin valset$ **then return** false
else $V := vset\ E := E - \{\langle (att, val), (att_1, val_1) \rangle \mid (att, val) \notin vset, (att_1, val_1) \notin valset\}$ **end if**
if graph G has a cycle **then return** false **else return** $plan$:= sequence of administrative requests corresponding to the topological sort of G;

to V_{ug}, it is checked that all the attributes in group are required in query, as no extra attributes are allowed in $\mathrm{RP}_{=}$ and deletion is not allowed. Line 4 – 8 creates edges in the graph, if a user group ug_1 is a negation conjunct to add another group ug_2 or ug_2 is a precondition for ug_1, then edge is drawn from ug_2 to ug_1, signifying that ug_2 should be added before ug_1. If cycles exists in the created graph then remove the cyclic paths and create topological sort on the remaining graph. The set of administrative requests based on the sort will provide the $plan_{ug}$ for user to groups assignment. Once the requests are executed in order, new effective groups are calculated for the user and computation continues from Algorithm 3.

Algorithm 3 extends algorithm defined in [107], which checks the final set of values required to satisfy the reachability query and find $\mathsf{canAddU}_{att}$ or $\mathsf{canAddUG}_{att}$ rules to add those values. Further to add the values in precondition of rules, it may in-turn need some other rules and values and so on. Therefore, algorithm traverses backward to find the set of values required to satisfy the query. Since the values can be achieved by user directly or from any of its effective groups, this backward search is done for user and all its effective groups as calculated by Algorithm 2.

The algorithm starts by checking if the query is satisfied in the current state, in that case empty plan is returned signifying that with only new group assignments query is satisfied. Otherwise, it creates a set of attribute value pair for values required in query q and also for current attributes of user and its effective groups (line 4-5). Line 6 checks if the union of current values of the user or its effective groups is not more than values required in q. If extra values are there, the algorithm returns false, as no delete rules are allowed. The algorithm calculates all positive precondition attribute value pairs required by user or its effective groups to get values in $toadd$ (line 7-17). Therefore, the final set of values required includes the values in query ($toadd$) and positive preconditions in user or its effective groups excluding their current values. Line 18 checks if rules exists to add all required attribute value pair or else returns false, as the values can not be added. Line 19-21 calculate negative conjuncts in rules required to add required values and returns false if the such values are present in current state. After passing through all checks, the algorithm starts creating a directed graph. Vertices (V) in the graph are attribute value pair of the values required in the query

q and the required positive preconditions excluding the values in the current state. Edges E will be drawn in the direction defined in the for loop (line 23-30). If the attribute value pair (att_1, val_1) is in the negative conjunct in administrative rule for (att_2, val_2) or (att_2, val_2) is a positive conjunct in a rule to add (att_1, val_1), the edge is created from (att_2, val_2) to (att_1, val_1). Since our query type is $\mathrm{RP}_{=}$, it requires an additional check so that no extra values are added to the user. Therefore, once the graph is created, we create a set $valset$, which includes values required in query and not present in the current state. If the created graph has vertex in $valset$ having incoming edge not from vertex in $valset$, algorithm returns false (line 32). Otherwise it removes all the edges from vertices not in $valset$. If cycles exists in the remaining graph then algorithm returns false, else the set of administrative request corresponding to the topological sort will return the $plan$.

Therefore, the overall reachability plan returned will be $plan_{ug}$ from Algorithm 2 and $plan$ from Algorithm 3.

Theorem 2. *Reachability query type* $\mathrm{RP}_{=}$ *for scheme* $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ *is P.*

Proof. Algorithm 2 and 3 describe the polynomial algorithms.

Complexity: The algorithm takes polynomial time to create directed graphs and then to compute topological sort. Its complexity is $\mathcal{O}(|\mathrm{UG}| \times |\mathsf{canAssign}| + ((\sum_{att \in \mathrm{UA}} |\mathrm{SCOPE}_{att}|) \times (|\mathsf{canAddU}_{\mathsf{att}}| + |\mathsf{canAddUG}_{att}| \times |\mathrm{UG}|)))$. □

In case of $\mathrm{RP}_{\supseteq}$ query type for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$, we remove the extra checks to verify if no extra values are present in current state (line 6). Further line 31-33 is not required as extra values are allowed to be added to user. With these minor changes, the complexity of $\mathrm{RP}_{\supseteq}$ for scheme $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ is P.

It should be noted that $\mathrm{RP}_{=}$ for $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ do no allow changes in group memberships of user. Therefore, computation for this scheme will start directly from Algorithm 3, obviating the execution of Algorithm 2. The $\mathrm{RP}_{\supseteq}$ query for $[\mathrm{rGURA}_{\mathrm{G}_1} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ will remove all extra conditions applied in Algorithm 3 for $\mathrm{RP}_{\supseteq}$ scheme for $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ as discussed above. Also, since $\mathrm{rGURA}_{\mathrm{G}_0}$ is a sub-problem of $\mathrm{rGURA}_{\mathrm{G}_1}$ we can conjecture that $\mathrm{RP}_{=}$ and $\mathrm{RP}_{\supseteq}$ for scheme $[\mathrm{rGURA}_{\mathrm{G}_0} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$ can be solved in polynomial time.

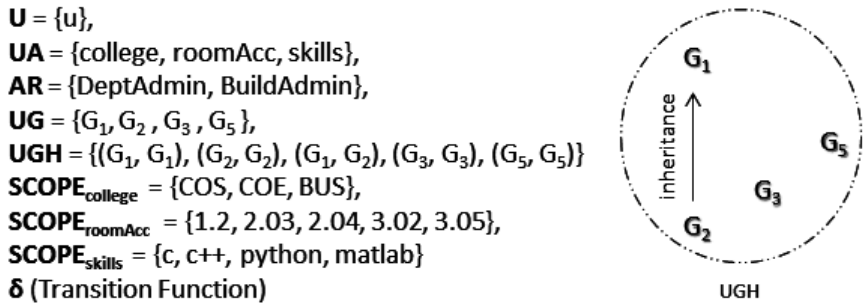


Figure 3.7: Input Starting State ($\gamma_0 \in \Gamma$)

3.5.3 Example Problem Instance

We will now illustrate the plan generation in two schemes discussed earlier with a sample input state and a set of reachability queries. Figure 3.7 defines the common input for both the schemes.

Plan Generation for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{N}}$]: Figure 3.8 shows attributes of user and groups along with user to group direct membership. Table 3.13 defines set of administrative rules allowed in scheme along with two reachability queries. We will first try to find a reachability plan (if exists) for query q_1 using Algorithm 1.

Initially, $plan$ is set to empty $\langle\rangle$. The initial state is checked to find if it has more attribute values than required in query q_1. In state γ_0, the effective values of user are e_roomAcc(u) = {2.04, 2.03, 3.02}, e_skills(u) = {c, c++}, e_college(u) = {COS}, which are all required in query. The while loop checks if query q_1 is satisfied in state γ_0, which is not true as some values are missing. Now algorithm starts adding new values to the user or its effective groups and also assign new groups to user based on administrative rules defined in Table 3.13. The first rule requires effective skills of user having value c++ and roomAcc attribute with value 2.04. to add 1.2 value to roomAcc by administrative role BuildAdmin. Since user satisfy these conditions and 1.2 value is not directly assigned in roomAcc(u) and the value is required in q_1, it adds 1.2 value to roomAcc(u). The administrative request add(BuildAdmin, u, roomAcc, 1.2) is also appended to the $plan$. The algorithm again goes through the while loop and checks if q_1 is satisfied. The user is still missing skills attribute value python. The algorithm now tries to add group G_3 to user u. The precondition of canAssign rule is satisfied by user, but the effective values for roomAcc attribute for group G_3

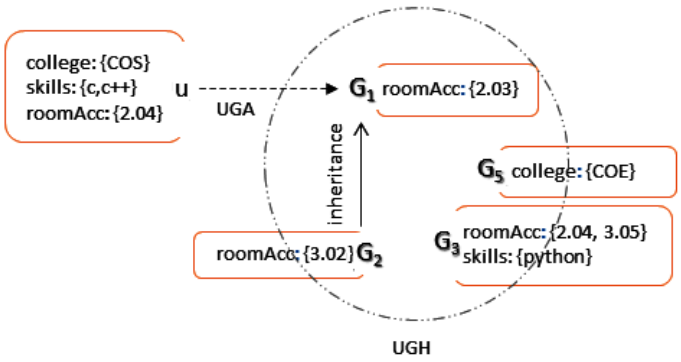


Figure 3.8: Initial State for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_{1+}}-\overline{\mathrm{N}}$]

Table 3.13: Example Problem Instance for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_{1+}}-\overline{\mathrm{N}}$]

Input: problem instance I = $\langle \gamma_0, q \rangle$ **Output:** *plan* or false
$\psi \in \Psi$:
$\mathsf{canAddU}_{\mathrm{roomAcc}}$ = {$\langle$BuildAdmin, c++ $\in$ e_skills(u) $\wedge$ 2.04 $\in$ roomAcc(u), 1.2$\rangle$ },
$\mathsf{canAddU}_{\mathrm{college}}$ = {$\langle$BuildAdmin, python $\in$ e_skills(u) $\wedge$ 3.05 $\in$ roomAcc(u), COE$\rangle$},
$\mathsf{canAddU}_{\mathrm{skills}}$ = {$\langle$DeptAdmin, c $\in$ e_skills(u), python$\rangle$},
$\mathsf{canAddUG}_{\mathrm{roomAcc}}$ = {$\langle$BuildAdmin, 3.02 $\in$ roomAcc(ug), 1.2$\rangle$ },
canAssign = {$\langle$DeptAdmin, $\mathrm{G}_1 \in$ directUg(u), $\mathrm{G}_3\rangle$ }
Queries:
$q_1 \in$ Q = {e_roomAcc(u) = {2.04, 2.03, 3.02, 1.2}, e_skills(u) = {c, c++, python}, e_college(u) = {COS}}
$q_2 \in$ Q = {e_roomAcc(u) = {2.04, 2.03, 3.02, 1.2}, e_skills(u) = {c, c++}, e_college(u) = {COS, COE}}

are {3.05, 2.04}, which is not the subset of values required in query. Hence, G_3 cannot be assigned to user u. Next, the algorithm checks rule for skills attribute to add value python and finds that preconditions to add value python are satisfied by user u. It appends corresponding request add(DeptAdmin, u, skills, python) to *plan* which results in total of two requests in the plan. The algorithm again checks new state against q_1 and finds the query is "strict" satisfied. It breaks while loop and returns plan = add(BuildAdmin, u, roomAcc, 1.2), add(DeptAdmin, u, skills, python).

We now check the satisfiability of query q_2 with the same initial state. Similar to q_1, query is checked against initial state to check extra values and value 1.2 for attribute roomAcc is added to the user and requests is appended to the plan. Second rule allows to add COE for attribute college, but the preconditions are not satisfied by user. We try to add group G_3 but it also adds extra values

which are not required in query. It can be noticed that after all the administrative rules are checked, the query cannot be satisfied and hence the algorithm returns false.

Plan Generation for $\mathrm{RP}_{=}$ in [$\mathrm{rGURA}_{\mathrm{G}_{1+}}$– $\overline{\mathrm{D}}$, SR_{d}]: Figure 3.9 shows user and group attributes along with user to group direct membership. Table 3.14 defines the set of administrative rules allowed in the scheme and three reachability queries. It should be noted that the rules in Ψ have negation conjuncts and single precondition with direct attributes or group memberships for each attribute value pair or user group. We will start with Algorithm 2 to assign new groups to the user. Once groups are assigned, attributes will be added to user or its newly computed effective groups. If Algorithm 2 doesn't add new groups, the computation will still be done by Algorithm 3 with old effective groups.

Algorithm 2 creates group assignment plan (defined as $plan_{ug}$) to assign new groups to user. Two administrative rules exists in canAssign relation. Since G_3 is not directly assigned to user u, precondition is satisfied and G_3 has value python for skill attribute, which is required in query q_1, algorithm adds G_3 to the set of vertices V_{ug}. Similarly group G_5 is also added to V_{ug}. There are no more canAssign rules, hence the algorithm starts adding edges to the graph. For (G_3, G_5) $\in V_{ug}$, since G_3 is a negation conjunct in precondition to add G_5, therefore, directed edge is drawn from G_5 to G_3. As here are no other relevant canAssign rules and vertices pair, it breaks the loop and creates a topological sort of the graph. Sort will have {G_5, G_3} order and the corresponding plan $plan_{ug}$:= assign($\mathrm{DeptAdmin}$, u, G_5), assign($\mathrm{DeptAdmin}$, u, G_3) is returned. Before proceeding to Algorithm 3, the request in $plan_{ug}$ must be executed to get new effective groups of the user. Algorithm 3 is used to assign attributes to user and newly computed effective groups (which will now have group G_5 and G_3 along with G_1 and G_2). It first checks if the query (q_1) is satisfied in the current state (line 2) which has new direct groups assigned using algorithm 2. Clearly query q_1 is satisfied with new group assignments only, hence the reachability plan for group assignments $plan_{ug}$ is returned.

For queries q_2 and q_3, group assignment plan $plan_{ug}$ is created similarly as above. Therefore, we will follow algorithm 3 with user's effective groups as G_1, G_2, G_3 and G_5. For q_2, current state

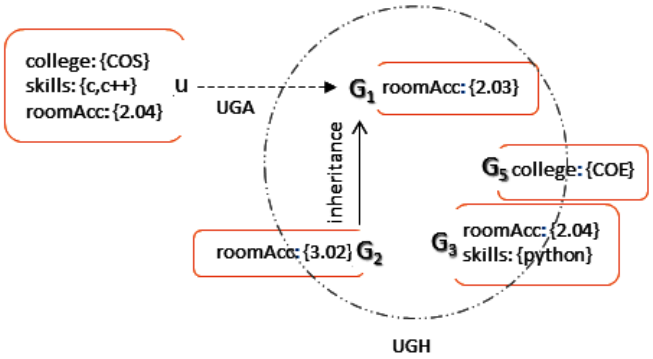


Figure 3.9: Initial State for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$

Table 3.14: Example Problem Instance for $\mathrm{RP}_{=}$ in $[\mathrm{rGURA}_{\mathrm{G}_{1+}} - \overline{\mathrm{D}}, \mathrm{SR}_{\mathrm{d}}]$

Input: problem instance I = $\langle \gamma_0, q \rangle$ **Output:** *plan* or false
$\psi \in \Psi$:
$\mathsf{canAddU}_{\mathrm{roomAcc}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{c{+}{+}} \in \mathrm{skills}(u) \wedge \neg(2.04 \in \mathrm{roomAcc}(u)),\ 1.2 \rangle\}$,
$\mathsf{canAddU}_{\mathrm{skills}} = \{\langle \mathrm{DeptAdmin},\ \mathrm{c} \in \mathrm{skills}(u),\ \mathrm{python} \rangle\}$,
$\mathsf{canAddU}_{\mathrm{college}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{matlab} \in \mathrm{skills}(u),\ \mathrm{BUS} \rangle\}$,
$\mathsf{canAddU}_{\mathrm{skills}} = \{\langle \mathrm{DeptAdmin},\ \mathrm{c} \in \mathrm{skills}(u) \wedge \mathrm{COS} \in \mathrm{college}(u),\ \mathrm{matlab} \rangle\}$
$\mathsf{canAddUG}_{\mathrm{college}} = \{\langle \mathrm{BuildAdmin},\ \mathrm{python} \in \mathrm{skills}(ug) \wedge \neg(2.04 \in \mathrm{roomAcc}(ug)),\ \mathrm{COE} \rangle\}$,
$\mathsf{canAssign} = \{\langle \mathrm{DeptAdmin},\ \mathrm{G}_1 \in \mathrm{directUg}(u),\ \mathrm{G}_3 \rangle, \langle \mathrm{DeptAdmin},\ \neg(\mathrm{G}_3 \in \mathrm{directUg}(u)),\ \mathrm{G}_5 \rangle\}$
Queries:
$q_1 \in \mathrm{Q} = \{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02\},\ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}\},\ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}\}\}$
$q_2 \in \mathrm{Q} = \{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02, 1.2\},\ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}\},\ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}\}\}$
$q_3 \in \mathrm{Q} = \{\mathrm{e\_roomAcc(u)} = \{2.04, 2.03, 3.02\},\ \mathrm{e\_skills(u)} = \{\mathrm{c}, \mathrm{c{+}{+}}, \mathrm{python}, \mathrm{matlab}\},\ \mathrm{e\_college(u)} = \{\mathrm{COS}, \mathrm{COE}, \mathrm{BUS}\}\}$

do not have value 1.2 for roomAcc attribute. The algorithm first computes *toadd*, which is the set of attribute value pair in q_2:

$$\begin{aligned} \mathit{toadd} = \{&\langle \mathrm{roomAcc},\ 2.04 \rangle, \langle \mathrm{roomAcc},\ 2.03 \rangle, \\ &\langle \mathrm{roomAcc},\ 3.02 \rangle, \langle \mathrm{roomAcc},\ 1.2 \rangle, \\ &\langle \mathrm{skills},\ \mathrm{c} \rangle, \langle \mathrm{skills},\ \mathrm{c{+}{+}} \rangle, \langle \mathrm{skills},\ \mathrm{python} \rangle, \\ &\langle \mathrm{college},\ \mathrm{COS} \rangle, \langle \mathrm{college},\ \mathrm{COE} \rangle\} \end{aligned}$$

It then calculates the current attribute value pair for user and its effective groups (Line 4-5):

$$\begin{aligned} \mathit{cur}_{\mathrm{u}} = \{&\langle \mathrm{roomAcc},\ 2.04 \rangle, \langle \mathrm{skills},\ \mathrm{c} \rangle, \langle \mathrm{skills},\ \mathrm{c{+}{+}} \rangle, \end{aligned}$$

$\langle \text{college},\ \text{COS} \rangle\}$

$cur_{\text{G}_1} = \{\langle \text{roomAcc},\ 2.03 \rangle\} \quad cur_{\text{G}_2} = \{\langle \text{roomAcc},\ 3.02 \rangle\}$

$cur_{\text{G}_3} = \{\langle \text{roomAcc},\ 2.04 \rangle\ , \langle \text{skills},\ \text{python} \rangle\}$

$cur_{\text{G}_5} = \{\langle \text{college},\ \text{COE} \rangle\}$

The algorithm checks if the current attributes of user and its effective groups are not extra than the values required in the query. Clearly, for query q_2, no extra values are present in current state. The algorithm next computes the positive conjuncts in the preconditions required to add the values in $toadd$. It first calculates for each attribute value pair in $toadd$ and then recalculates for each positive preconditions attribute value pair also. For example, positive conjunct for user to add $\langle \text{roomAcc},\ 1.2 \rangle$ in $toadd$ is $\langle \text{skills},\ \text{c++} \rangle$ and for $\langle \text{skills},\ \text{python} \rangle$ in $toadd$ is $\langle \text{skills},\ \text{c} \rangle$. Therefore (using line 9), $ppre'_u$:= $\{\langle \text{skills},\ \text{c++} \rangle, \langle \text{skills},\ \text{c} \rangle\}$. It then recomputes $ppre_u$ by combining its values with newly computed $ppre'_u$ after removing values already present in $ppre_u$ or cur_u. In this case, no new value is added in $ppre_u$, as both the values in $ppre'_u$ are already present in cur_u. Similarly, the positive preconditions are calculated for each effective groups.

$ppre_{\text{u}} = \{\}, \quad ppre_{\text{G}_1} = ppre_{\text{G}_2} = ppre_{\text{G}_5} = \{\langle \text{skills},\ \text{python} \rangle\}$

$ppre_{\text{G}_3} = \{\}$

Next, in line 18, the algorithm checks if rules exists for values required in $toadd$ and positive preconditions excluding the current values. Clearly, rule exists for $\langle \text{roomAcc},\ 1.2 \rangle$ pair and all other values are already present in user or its effective groups. It then calculates negative conjuncts for user and its effective groups in preconditions to add values in $toadd$ and positive preconditions excluding current state. For user, $\langle \text{roomAcc},\ 1.2 \rangle$ has negation conjunct $\langle \text{roomAcc},\ 2.04 \rangle$ in $\mathsf{canAddU}_{\text{roomAcc}}$. Remaining negation conjuncts are as follows:

$npre_{\text{u}} = \{\langle \text{roomAcc},\ 2.04 \rangle\}$

$npre_{\text{G}_1} = npre_{\text{G}_2} = npre_{\text{G}_3} = \{\langle \text{roomAcc},\ 2.04 \rangle\}$

$npre_{\mathrm{G}_5}$ = {}

Line 21 checks if the negation conjuncts exists in current values of either user or its effective groups. User has $\{\langle \mathrm{roomAcc},\ 2.04\rangle\}$ pair in cur_u, therefore, roomAcc attribute cannot get value 1.2 required in q_2 since only single rule exists for user or groups. Hence, the algorithm returns false for query q_2. For query q_3, $toadd$ values are:

$toadd$ = $\{\langle \mathrm{roomAcc},\ 2.04\rangle\ , \langle \mathrm{roomAcc},\ 2.03\rangle,$
$\langle \mathrm{roomAcc},\ 3.02\rangle\ , \langle \mathrm{skills},\ \mathrm{c}\rangle, \langle \mathrm{skills},\ \mathrm{c{+}{+}}\rangle,$
$\langle \mathrm{skills},\ \mathrm{python}\rangle, \langle \mathrm{skills},\ \mathrm{matlab}\rangle$
$\langle \mathrm{college},\ \mathrm{COS}\rangle, \langle \mathrm{college},\ \mathrm{COE}\rangle, \langle \mathrm{college},\ \mathrm{BUS}\rangle\}$

The current values are still the same as defined in query q_2. The algorithm calculates the positive conjuncts in preconditions as:

$ppre_{\mathrm{u}}$ = $\{\langle \mathrm{skills},\ \mathrm{matlab}\rangle\}$

$ppre_{\mathrm{G}_1}$ = $ppre_{\mathrm{G}_2}$ = $ppre_{\mathrm{G}_5}$ = $\{\langle \mathrm{skills},\ \mathrm{python}\rangle\}$

$ppre_{\mathrm{G}_3}$ = {}

The negation conjuncts are calculated as:

$npre_{\mathrm{u}}$ = {}, $npre_{\mathrm{G}_1}$ = $npre_{\mathrm{G}_2}$ = $npre_{\mathrm{G}_3}$ = $\{\langle \mathrm{roomAcc},\ 2.04\rangle\}$

$npre_{\mathrm{G}_5}$ = {}

These negation values are not present in user or all of its effective groups. Therefore, the algorithm creates directed graph (line 22-30) with vertices V := $\{\langle \mathrm{skills},\ \mathrm{matlab}\rangle\}$, $\{\langle \mathrm{college},\ \mathrm{BUS}\rangle\}$. Edge in E is drawn from $\{\langle \mathrm{skills},\ \mathrm{matlab}\rangle\}$ to $\{\langle \mathrm{college},\ \mathrm{BUS}\rangle\}$ as $\{\langle \mathrm{skills},\ \mathrm{matlab}\rangle\}$ is a precondition conjunct in rule to add $\{\langle \mathrm{college},\ \mathrm{BUS}\rangle\}$. Line 31 calculates $valset$ which in this case is same as V. Since no cycle exists in the graph, topological sort is created. The final reachability plan to satisfy the query q_3 is $plan$:= assign(DeptAdmin, u, G_5), assign(DeptAdmin, u, G_3), add(DeptAdmin, u, skills, matlab), add(BuildAdmin, u, college, BUS). The administrative requests must be executed as ordered in the reachability plan.

CHAPTER 4: ACCESS CONTROL FOR SMART CONNECTED CARS

In this chapter, we discuss the access control requirements in cloud assisted smart cars and propose an authorization framework. We also present dynamic groups and attribute based secure communication and data exchange solution for connected vehicles ecosystem. Significant portion of this work has been presented at the following venue [89]:

- Maanak Gupta and Ravi Sandhu, Authorization Framework for Secure Cloud Assisted Connected Cars and Vehicular Internet of Things. In Proceedings of the 23rd ACM Symposium on Access Control Models and Technologies (SACMAT), June 13-15, 2018, Indianapolis, Indiana, pages 193-204.

4.1 Motivation and Scope

Smart cities and intelligent transportation has been a vision of future society. IoT plays an important role to make transportation smarter by introducing connected cars and vehicular communication. Smart cars ecosystem involves interaction and V2X data/messages exchange between several entities including vehicle to vehicle (V2V), vehicle to road infrastructure (V2I), vehicle to human (V2H), intravehicle, and vehicle to cloud (V2C). Vehicular Ad-hoc Networks (VANETs) provide necessary connectivity which is extended with use of smarter devices and cloud or fog infrastructures. Sensors in and around connected car 'talk' to each other for smarter decisions and convenient transportation experience to user. Our vision of smart vehicles ecosystem harnesses computation and storage capabilities of cloud and the concept of virtual objects (e.g. AWS shadows [42]).

Security and privacy have been a serious concern and challenge for the adoption of IoT. The gravity of these issues is magnified when we think about implications in vehicular IoT and the emerging concept of autonomous cars. This ecosystem has connected cars as its most important, and also most vulnerable, entity. With over 100 millions lines of code, more than 100 electronic control units (ECUs) and broad attack surface opened by features such as on-board diagnostics, driver assistance systems and airbags, it becomes a challenge to protect this smart entity. Fur-

ther, the communication among smart objects (vehicle to vehicle, vehicle to infrastructure etc.), mobility, and dynamic network topology makes it even harder to secure the system.

To address security and privacy issues, we first propose an extended access control oriented architecture (E-ACO) for connected vehicles ecosystem, which is an extension to the recently proposed ACO architecture for IoT [37]. The prime difference between these architectures is the introduction of clustered objects, which are objects with multiple sensors, and possible interaction between sensors in same clustered object or between different object sensors. Clustered objects are particularly relevant in case of connected cars, traffic lights or other smart devices which have multiple sensors and ECUs mounted on them. This authorization framework illustrates different interaction and data exchange scenarios in vehicular IoT and proposes access control models at various E-ACO layers including physical objects, virtual objects, cloud layer and applications.

Secondly, we propose a model, referred as CV-ABAC$_{\mathrm{G}}$, for data exchange and resource access using secure cloud based communication in smart vehicles ecosystem. This model takes into account the user-centric privacy preferences along with system-defined policies to make access decisions. In this work, we present a novel concept of groups in context of smart cars, which are dynamically assigned to moving entities like vehicles, based on their current GPS coordinates, direction or other attributes, to ensure relevance of location and time sensitive notification services offered to drivers, provide administrative benefits to manage large numbers of entities and enable attributes inheritance for fine grained authorization.

4.2 Cloud Assisted Vehicular Internet of Things

The vision of intelligent transportation encompasses connected cars and vehicular IoT as an important component. The eventual goal of smart city ecosystem is the integration of vehicles, infrastructure, smart things, homes or ultimately any thing to promote efficient transportation, accidental safety, fuel efficiency etc. and for pleasant travel experience to the driver. The technology involves communication between vehicles (V2V), vehicle to human (V2H), vehicle to cloud (V2C), vehicle to infrastructure (V2I) etc. to exchange vehicle telematics [39, 71] and gather information

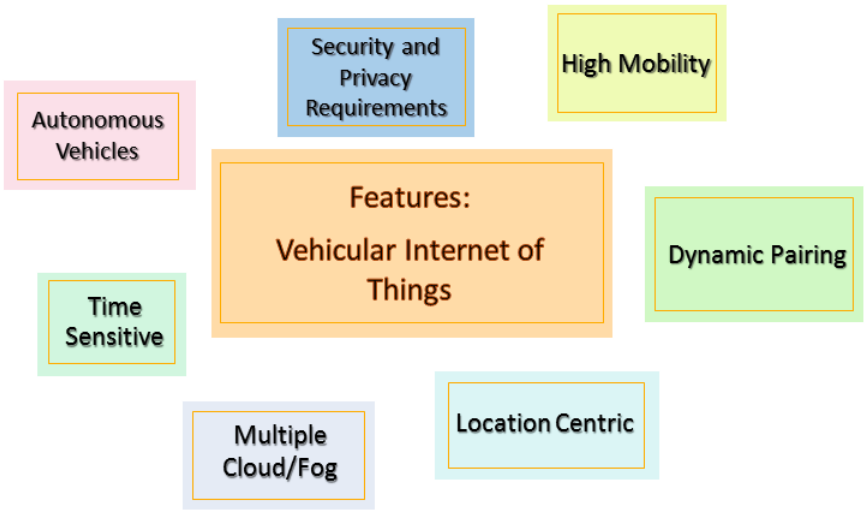


Figure 4.1: Vehicular IoT Distinguishing Characteristics

about surroundings to offer services to the users. Safety applications in IoV require basic safety messages (BSMs) to be exchanged among smart entities, which contain information about vehicle position, heading, speed, etc, related to vehicle state and predicted path [16]. Such interaction can happen using dedicated short-range communications (DSRC) technology (similar to WiFi, secure and reliable) which allows rapid communications (up to 10 times per second) between elements of vehicular IoT network required for end user applications.

4.2.1 Characteristics and Cloud Architectures

Vehicular IoT inherits intrinsic IoT characteristics of data sharing, communication and accumulation in cloud. However, dynamic topology structures, dynamic communication, mobility, network scale, and non-uniform nodes distribution (shown in Figure 4.1) are some features that distinguish it from other IoT domains, resulting in new security and privacy challenges. Further, several applications in IoV domain are very time and location sensitive; for example, BSM information about traffic congestion from a neighbouring vehicle or a traffic light, or about ice on bridge or an accident report to a nearby hospital etc. makes IoV ecosystem very dynamic. Internet of Vehicles involve different kinds of objects (as shown in Figure 4.2) based on their mobility, functionalities or processing capabilities. Some smart objects are static in nature; for example, beacons

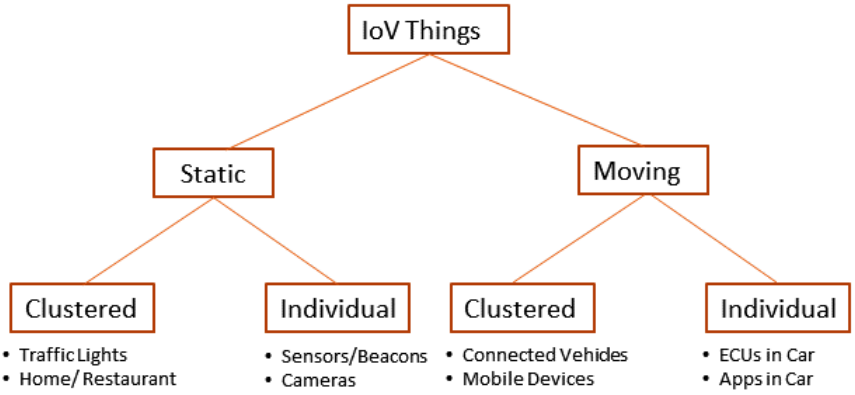


Figure 4.2: Smart Object Types in Connected Vehicles Ecosystem

outside a restaurant, or sensor on a smart traffic light whereas moving objects include connected cars, pedestrian with mobile phones, etc. Further, some of these are individual objects with single sensor performing only one function whereas some are clustered objects having multiple sensors associated with them. A connected car has several ECUs and sensors on it, and hence is referred as a clustered object whereas a single ECU in a car generally performs one function and is an individual object. Such characterization is necessary as it drives our access control framework and models. Several applications of connected cars and vehicular IoT are envisioned for smart city intelligent transportation initiative, including the following.

- **Safety and Assistance:** With machine to machine (M2M) communication among vehicles and infrastructure, these applications provide real-time information about other vehicles and traffic to control speed, in-lane position control or road work warning from signboards. Further, during inclement weather, under non-ideal driving conditions, or blind spots, even pedestrian with mobile phones can automatically exchange safety messages about their position or speed with incoming vehicle such as while crossing roads to alert drivers.

- **Diagnostic and Maintenance:** Remote diagnostic and predictive maintenance of vehicles through manufacturer or authorized mechanic will save time and money. Vehicle sensor data can be send to cloud for processing to predict vehicle mechanical issues. Over the air (OTA) updates can also be issued by manufacturer for fixing car firmware which will obviate the need to go to mechanic. Fleet management applications provide real-time telematics, driver

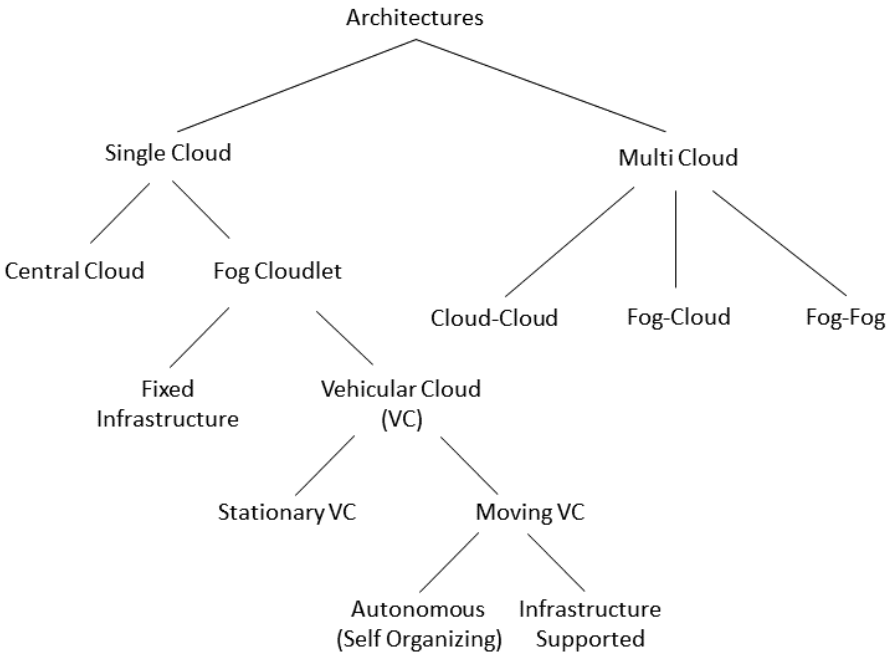


Figure 4.3: Different Cloud and Fog Cloudlet Architectures in Vehicular IoT

fatigue detection and package tracking.

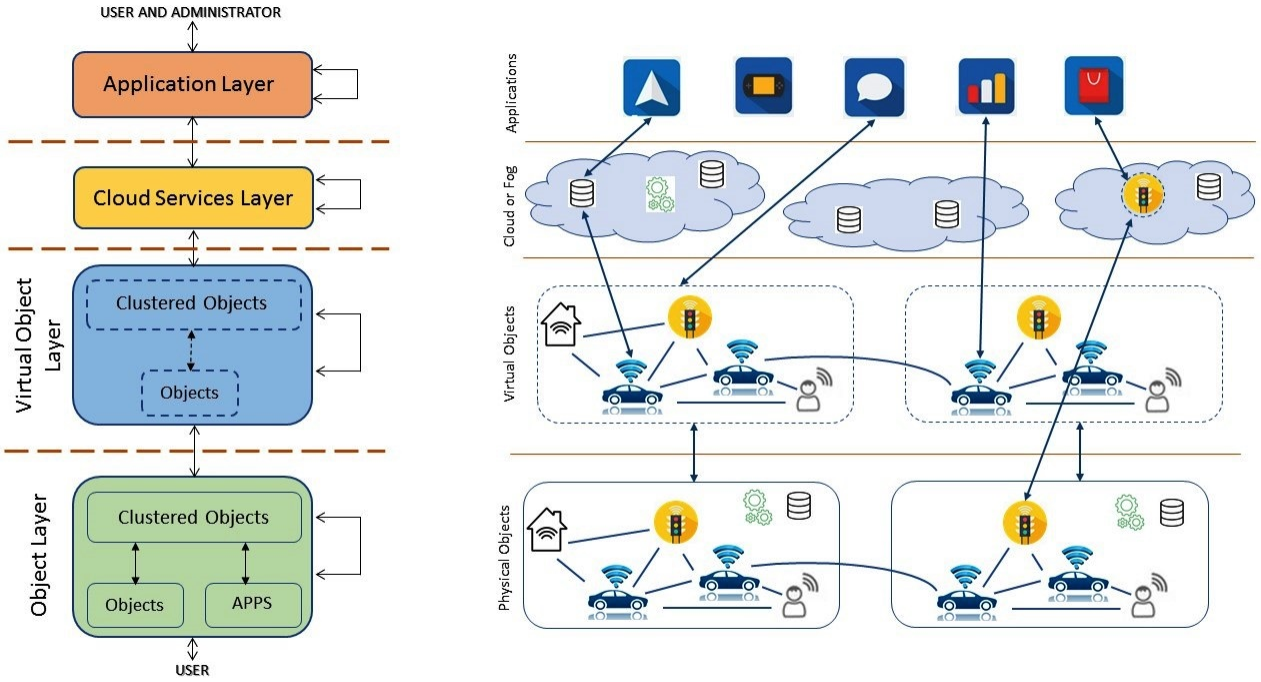
- **Information and Entertainment:** Driving based insurance models have been introduced which will assess the driver behaviour to determine insurance premiums. Real-time parking information can be shared between parking garages and vehicles. Restaurant and gas stations can send offers to nearby vehicle's dashboard. Car-pooling, connected driving [110], web-browsing, music etc. are some additional connected vehicles and smart car applications.

We believe cloud platforms like Amazon AWS, Microsoft Azure etc. will play an important role to fully harness the potential and applications of connected vehicles. Further, the use of edge or fog computing [51] is imperative to resolve the issues of high latency, low bandwidth and communication delays pertinent to using central cloud, which are very critical in time and location sensitive IoV applications [19]. Figure 4.3 shows various single and multi cloud scenarios viable in vehicular IoT. Single cloud architectures may involve only one central cloud which manages user applications, virtual objects and data generated from smart entities in a wide geographic area, such as a city. This architecture is not feasible because of latency and other issues mentioned above. Fog or edge computing is essential, and we believe IoV can either use vehicular cloud (i.e

the resources offered by smart vehicles and infrastructure on road) or a fixed infrastructure setup along the road where compute and storage clusters are dedicated for small areas. It is also possible to use fog structure for each connected vehicle, and sensors in the vehicle have virtual objects in the fog which can be used to enable intra-vehicle communication. Vehicular cloud (VC) can be stationary where the vehicles are standing in a mall parking lot and offering their resources for an incentive (like free parking) or moving VC where vehicles while moving may form cloud using broker [79, 80, 101, 131, 180] and can leave or join the cloud if in specified geographic range. Further, these moving VCs can be supported by fixed infrastructure (example, a traffic light on the roadside acting as a broker) or moving vehicles in autonomous manner can form a VC. In multi-cloud IoV architectures, we envision to have either multiple clouds, cloud-fog or multiple fogs setup. However, we assume a central cloud and multiple fog architectures are a good fit to cover most connected car and IoV applications, as discussed later in our extended ACO architecture.

4.2.2 Extended ACO Architecture

Connected Cars and Vehicular IoT ecosystem has several heterogenous devices (individual or clustered) and in-built car applications which cooperate to provide services to the end users. Some devices are independent (camera on street or beacons on restaurant) whereas some belong to a larger clustered object (ECU or sensor in a connected car). Hence, we propose to incorporate this distinction into previously defined access control oriented architecture (ACO) [37] to address connected vehicles ecosystem access control requirements. An important reason to incorporate clustered objects is to reflect cross-vehicle and intra-vehicle communication. The fact that two connected cars can exchange basic safety messages (BSM) with each other reflects clustered object communication. Such concept is not defined in ACO architecture which is proposed for generic IoT systems. Besides objects, these clustered objects may have applications running in them; for example, a car may have a navigation application installed in it, or a safety warning application, which may interact with sensors on a smart sign-board to warn the driver via car dashboard or seat vibration or buzzer. It should be noted that these sensors or applications may access sensors in car they belong



a) Extended ACO Architecture for Connected Car and IoV

b) Connected Car and Vehicular IoT Components in Extended ACO Layers

Figure 4.4: Extended ACO Architecture for Connected Cars and Vehicular IoT

to or possibly sensors on other cars also.

Figure 4.4 shows our proposed extended ACO (E-ACO) architecture along with the corresponding vehicular IoT components at different E-ACO layers in Figure 4.4 (b). E-ACO architecture has four layers similar to ACO: Object layer, Virtual Object layer, Cloud Services layer and Application layer, where the communication can happen within a layer (shown as self loop in Figure 4.4 (a)) and the adjacent layers above and below. We will now discuss layers in more detail.

Object Layer: The object layer introduces clustered objects which have multiple individual sensors or smart objects. The clustered objects may also have several built-in applications (like tire-pressure monitoring) installed within them. These applications can communicate with ECUs and sensors in same car (or neighbouring car) to get data and update information to the drivers. Some of these applications accumulate data and send it to the cloud infrastructure for further analysis; for example diagnostic applications installed by the manufacturer which will collect data from critical engine sensors and send to the cloud for processing and offering customers with OTA maintenance services. The in-vehicle communication for applications, ECUs and sensors is

supported by different networking technologies including Controller Area Network (CAN), Local Interconnect Network (LIN), Ethernet, Media Oriented Systems Transport (MOST) etc. Communication can occur between objects (and clustered objects) in the object layer and also with the layers above (virtual object) and below it (user). Communication across objects (within the object layer) among different vehicles or clustered objects is feasible via technologies like dedicated short-range communications (DSRC), Bluetooth, WiFi, and LTE. An example interaction in object layer is BMW connect application in phone which reads address from phone and send to the car navigation system, or V2V BSM exchange using DSRC.

It should be noted that instead of introducing clustered objects as a separate layer in E-ACO, we have added them to the same object layer of ACO architecture, which reflects the binding between objects, applications and the clustered object to which they belong. We believe the relationship between objects and clustered objects is important, for example, a lane departure sensor in car will have some attributes (like vehicle id) it inherits from the car and such binding is shown by putting them in same layer. These clustered objects and cars also have applications associated with them which offer services to drivers inside. For example, a rear vision system is an application in cars to get rear-view, which gets data from rear-camera (an object) to provide dashboard view to the driver. Other applications, including tire-pressure monitoring which 'talks' to sensor installed in tire, cabin monitoring system, info-tainment systems etc, are in-built in connected cars and can communicate with sensors or other applications in system. These applications in object layer of E-ACO are an add-on to the object layer in ACO architecture and reflect its importance in intelligent transportation ecosystem which is dependent on in-built applications supported by smart cars.

Virtual Object Layer: Communication of sensors, vehicles and other smart entities may also involve virtual objects or cyber entities to eliminate connectivity, heterogeneity and locality issues. The most important smart entity in IoV, a smart car, when in motion will pass through areas with low or no internet connectivity. In such scenario, it is imperative to create a cyber entity of smart car (and its sensors) in the cloud so that the last state and desired state information of car (and sensors) can be sent to the virtual entity when car is not connected. Once the physical object gets back

internet connectivity, the virtual entity will push information/state to its physical counter part. For example, if a problem is diagnosed in powertrain control ECU of a car and a command needs to be sent by mechanic to ECU to control air-fuel ratio. In case a car has internet connectivity, message can be sent directly to ECU, but if there is no connectivity message should be sent to virtual entity of the ECU which will push message to physical ECU when car gets connectivity and syncs virtual and physical entity. We envision the virtual object layer in E-ACO architecture will have one or many cyber entities (virtual objects or device shadows) for both clustered and individual objects. Physical objects can communicate with their cyber counterpart using HTTP, MQTT, AMQP or CoAP protocols. When sensors s_1 and s_2 across different vehicles or clustered objects communicate with each other, the sequence of communication via virtual object layer should follow starting s_1 to vs_1 (virtual entity of s_1), vs_1 to vs_2 and vs_2 to physical sensor s_2. Similar communication can be envisioned for in-built car applications which can indirectly exchange information from physical sensors through their cyber counterparts created in cloud, vehicular cloud or fog architecture. It is possible to create a fog cloudlet for each vehicle where cyber entities will reside and support the indirect communication within physical sensors and ECUs inside car. Our E-ACO architecture does not support cyber-entity for in-car applications supported by IoV and will not create virtual objects for such applications [1].

Cloud Services and Application Layer: Since most user smart cars applications are cloud supported (i.e. use cloud infrastructure and services), we explain them together to provide a better understanding of these two mutually dependent E-ACO layers. Cloud layer provides storage and processing whereas application layer provides application interface to users to control and interact with object layer components as discussed in ACO architecture [37]. Over the air (OTA) updates for firmware and other software components in the cars are through the cloud service layer where only authorized users are allowed to issue OTA. User and applications can access the data pushed into the cloud by smart infrastructures for offering value added services to customers. Our proposed architecture assumes to have both central cloud and fog (instantiated by vehicular cloud)

[1]Note that Amazon AWS IoT does allow applications to have a thing shadow [42] but comprehensive smart cars use-cases to support the functionality are still missing in literature.

component in IoV ecosystem but are collectively represented as cloud services. An important use for cloud layer in IoV and connected cars involves defining security policies for authorized vehicular communication, which we understand is missing in literature. Further, we assume that virtual entities of various objects can be created in both central cloud and fog depending on the use-cases and the scope of applications which are accessing the objects. For example, an accidental safety application will have limited geographic scope and hence will access virtual objects created in fog (to overcome latency issues); whereas, a health-monitoring application may access body sensors via virtual objects created in central cloud. Cloud services and applications can access information and data from virtual objects using MQTT or other relevant protocols. It should be noted that most IoV architectures and use-cases we studied [61, 80, 110] don't have virtual object layer and include only object, cloud services and application layer. Communication between cars, sensors and applications in object layer do not involve virtual objects and is done using lower layer protocols like DSRC, WAVE, Bluetooth or WiFi. Sensors can directly send data to cloud storage for processing without involving virtual objects, which is then used by applications. However, connectivity issues in moving cars and communication heterogeneity among entities supports the need for virtual layer, as discussed earlier in this section.

Figure 4.4 (b) shows an instance of connected vehicles ecosystem with physical objects (like smart cars or traffic lights) along with their cyber counterparts in virtual objects layer, and other E-ACO layers. It can be seen that physical objects communicate with virtual objects, and user applications are accessing data through cloud which is pushed by virtual entity of an object. Storage and processing icons at object layer symbolizes road-side infrastructures which can also help to store data from vehicles and filter before pushing data to the central cloud. Virtual objects are created at both fog and central cloud to satisfy different application needs.

4.3 Authorization Framework for Smart Cars Ecosystem

The dynamic and distributed nature of vehicular IoT brings in challenges to secure the ecosystem. Broad attack surface and numerous external interfaces along with the intrinsic characteristics of

IoV makes it hard to ensure security and privacy of the components and data inside. Access controls are important to restrict unauthorized access to data, sensors, applications, infrastructure and other resources in connected cars and IoV. Applications like MobEyes [119] and CarSpeak [116] allow vehicles (or sensors) to access not only its own sensors but also neighbouring vehicle sensors to get data and information. The exchange of BSM messages among vehicles and smart entities, and their use must be trusted and checked. Further, in-vehicle communication along buses between ECUs and applications should be secured. Such exchange must be authorized to ensure confidentiality and integrity of vehicle's and user's personal data, and to prevent remote (or physical) control of connected smart entities. In this section, we define an access control framework that reflects authorization needs at various layers of extended ACO architecture discussed earlier.

Several interaction scenarios exist in vehicular IoT ecosystem which makes it hard to comprehend different access control decision and enforcement points, together with other security requirements. Based on the extended ACO architecture, we have put together various vehicular IoT communications into three categories: Object Level, Virtual Object Level and Cloud Services Level as shown in Figure 4.5. Since most user applications are cloud based which use services and resources in cloud, we have bundled the interaction of IoV entities with cloud and applications together. As discussed in the E-ACO architecture, each layer components interact with themselves (components in same layer) and the components in layers immediately above and below it. Two types of interactions exist in E-ACO, direct and indirect, marked with solid and dashed boxes shown in figure. Communication between adjacent and same layer is direct communication whereas indirect includes interaction beyond adjacent layers i.e. two or more layers above or down in E-ACO. For example, interaction between clustered object and objects inside the clustered object is direct, as they belong to the same object layer whereas interaction between an application in application layer and object will be indirect as applications will interact with object via its virtual entity created in cloud. It is possible to have interactions overlapping in two categories, e.g., cloud service (CSR) and virtual object (VOB) interaction is part of both cloud services and virtual object category. Following are the authorization framework categories and IoV communication scenarios.

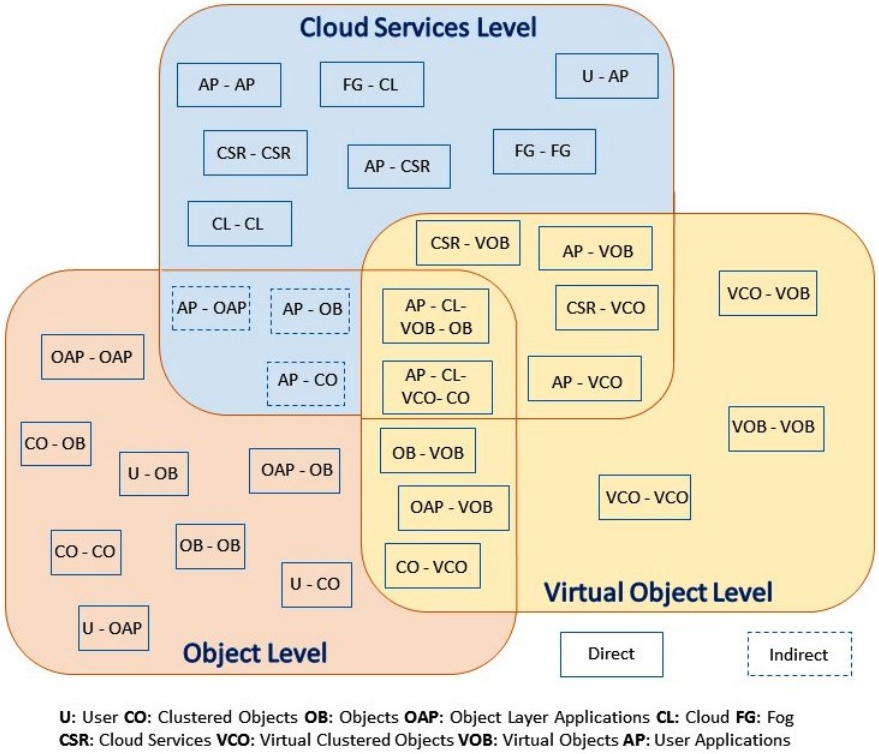


Figure 4.5: Different Interactions in Vehicular IoT Ecosystem

- **Object Level:** This category covers object layer interaction within itself and with adjacent layers (virtual objects and users) in E-ACO architecture. Some interaction types (shown in Figure 4.5) include between clustered objects (CO-CO), between clustered object and object (CO-OB) for example smartphone and car USB port, between user and sensors (U-OB), between sensor and any application running inside car (OB-OAP), and between ECUs (OB-OB). Access control models to authorize each of these interactions and resulting data exchange are required. BSM exchanges between connected cars using DSRC is an example communication that needs entity authorization to ensure integrity of message, which must be addressed by appropriate access control methods.

- **Virtual Object Level:** This includes communication of virtual entities with real objects, with cloud services or with user applications. Some examples include, between virtual objects (VCO-VCO, VOB-VOB), between application and virtual objects (AP-VOB), cloud services and virtual objects (CSR-VOC, CSR-VOB) etc. Most of these communications are

through publish-subscribe protocols like MQTT, DDS or through HTTP, CoAP. Recently, Alsehri and Sandhu [38] presented access control models for VOB-VOB interaction in topic based communication using CapBAC (Capability based access control), ACLs and ABAC.

- **Cloud Services Level:** Cloud provides necessary storage, processing and services to unleash true IoT potential. Further, most applications are also cloud based with their software and hardware components supported in cloud. Therefore, this category includes both application and cloud interactions with IoV entities and virtual objects. The layer also considers multi-cloud or fog-cloud interactions which are important in distributed IoV. Some interactions in this category (shown in Figure 4.5) include: between user application and cloud services (AP-CSR), multi-cloud or fog interaction (CL-CL, FG-CL), indirect interaction between application and objects (AP-OB), cloud services (CSR-CSR) etc.

In-vehicle network allows interaction among sensors and applications inside the car, which also needs protection. Such communication can fit into above categories depending if entities involved are physical, virtual or applications. CAN bus and other intra-vehicle communication can be protected by assigning ACLs and capabilities to ECUs to prevent spoofing and other attacks. TCUs (Telematics Control Units) or Gateways have been used to separate critical ECUs from non-important subnetworks and also act as a common external interface to connected car. Access control models should be developed for various interactions in each category to control communication and data exchange. Note that our authorization framework does not include physical tampering and OBD port connectivity. In next section, we discuss some access control approaches relevant to fit in the vehicular IoT authorization framework.

4.4 Access Control Approaches

Researchers have investigated authorization requirements in IoT systems and proposed several access control models and implementations [55, 90, 92, 95, 111, 134, 151, 184]. Recently, an access control model for virtual objects was proposed [38] using ACLs, CapABAC and ABAC. AWS

access control model for IoT is discussed [49] which uses policies to control physical, virtual, cloud services and other communications in a publish subscribe exchange protocol.

We conclude that intelligent transportation environment requires access control policy decision and enforcement at two levels: external communication and in-vehicle internal communication. Access control for external interface will secure authorized access to vehicle's data, sensors, ECUs and applications from external entities (like vehicle, traffic light, smartphones or user applications etc.) whereas internal mechanisms will secure ECUs and in-car applications communication and data exchange in a connected car supported by CAN bus, Bluetooth, WiFi etc. Securing external interface may not be enough to stop hackers, as they could impersonate a trusted device and by-pass external access control. Also, in case if some ECUs with external interface are compromised, second level access control will protect critical systems in connected car. Vehicles discover new vehicles and infrastructure and start exchanging BSMs with them. Vehicular IoT mainly has two data exchange scenarios: static and dynamic, where static considers interaction due to long lasting relation for example, vehicle and owner or car manufacturer. Dynamic communication is temporary and occurs when entities are at certain place, or in geographic range with no prior relation between them. Also, static relation may share more private information which might not be the case in dynamic relation. These relations can help understand and develop access controls in the Internet of Vehicles. Another approach may require multi-layered access control where the type of action required on an object determines the authority who can take access decision. For example, controlling an autonomous vehicle may require permissions from both owner and transportation authority, whereas reading data from vehicle may only need owner's consent.

We believe that clustered objects are important in access control decisions and can help to make preliminary decision. In case of vehicles, it is not only the vehicles which share BSMs, but also the sensors or applications in them which communicate. Therefore, first level check will ensure if two vehicles (as clustered objects) are allowed to talk, without considering their in-built sensors. If authorized at first level, next level access control will include sensors, applications and ECUs of the vehicles to make the final decision. Concept of trust can be introduced where only trusted

entities can communicate. Trust can be established based on interaction, or relationships among two entities. For example, entities who have exchanged data earlier are more trusted; home and vehicle belonging to same owner are more trusted and can communicate. PKI based trust establishment in Security Credential and Management System (SCMS) [174] supported by USDOT is an important system to ensure BSM confidentiality and integrity in V2V communication. Further, attribute based [105] solutions can be added where IoV entities can inherit set of attributes from their geographic location, or from manufacturer. In such cases, attribute based policies can be used to determine sensors communication after trust is checked between their vehicles. Attributes which can be used in access decisions include geographic position, current speed, acceleration, deceleration, road surface temperature or other vehicle telemetry. Two level policy may be required: one at cloud level (to control V2V, V2I like communication) and another at fog or vehicular cloud level (to control intra-vehicle ECU's, applications or sensors interaction). Both single and multi-cloud scenarios can exist in which vehicles in same or across clouds can interact, which will also require access controls. Administrative models [38, 88] are also needed to support administration of vehicular IoT operational access control models.

4.5 Cloud Assisted Real-World Use Cases

In this section, we will discuss some use-cases in relevance to our authorization framework, incorporating the extended ACO architecture (shown in Figure 4.4), various smart vehicles communication exchange scenarios (shown in Figure 4.5), using cloud and fog architectures, and entities of vehicular IoT ecosystem as shown in Figure 4.6. We have classified our use-cases into single cloud and multi cloud systems to reflect local or global scope of entity communications and user IoV applications, however, all applications in single cloud can be extended to multi-cloud and vice versa. Our prime objective is to describe how interactions and data exchange takes place in distributed and dynamic vehicular IoT ecosystem and to elaborate on various access control decision, and enforcement points requirements.

Most applications in vehicular IoT are time and location sensitive, which require real time

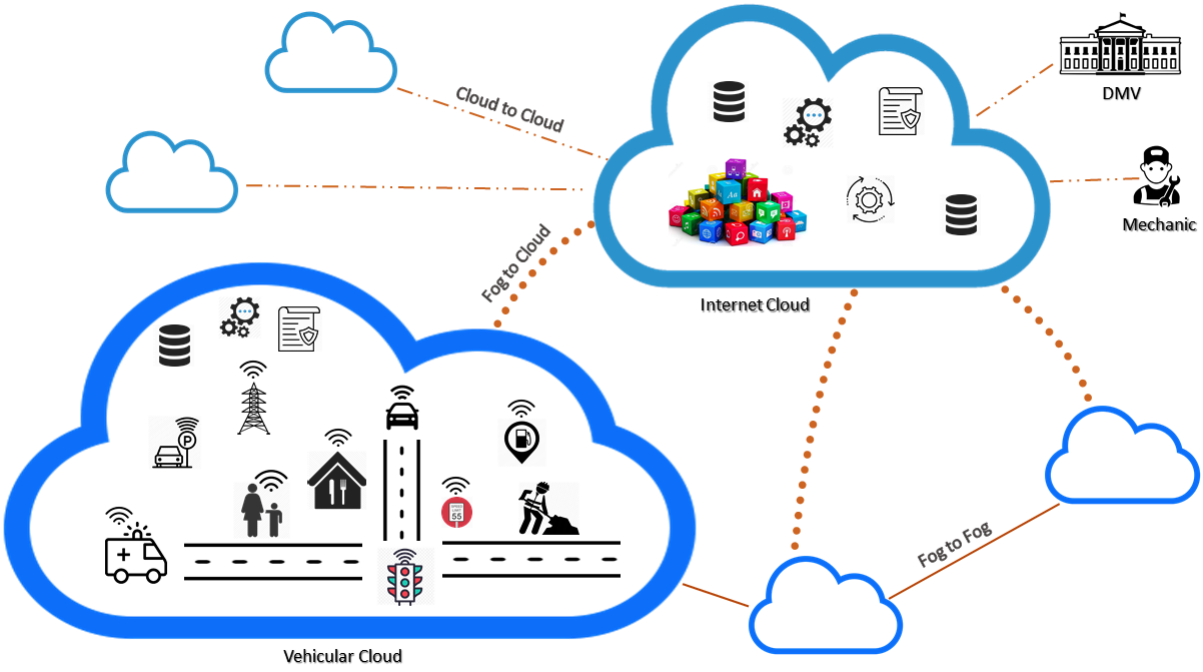


Figure 4.6: Connected Cars and Internet of Vehicles Ecosystem

processing of information gathered from smart vehicles, sensors, ECUs and other smart objects present in a limited geographic area. To resolve issues related to latency and bandwidth pertinent to using central cloud, we believe vehicular cloud (VC) will play an important role, where the storage and computation present in smart vehicles or road side infrastructures (smart traffic lights, sign boards etc.) can be used to support IoV applications. Hence, our single cloud applications are supported by fog or cloudlet instance in the form of a VC where physical objects will have their cyber counterpart (virtual objects). It is also possible to have a fog instance for each connected car and any communication within the car is supported through it. Other scenarios may need to have multiple virtual objects for a single physical object where some objects are in VC and some in central internet cloud, required for more persistent state or for non-time sensitive applications or where the interacting IoV entities are not present in the range of same vehicular cloud. Such use-cases are discussed in multi cloud or fog-cloud architecture scenarios.

4.5.1 Single Cloud System

Single cloud applications include entities in limited geographic area communicating and exchanging information. A pedestrian crossing a road sends an alert message to an approaching car, or remote parking capability in BMW 7 series assists driver to park car using touch screen key are some examples of short-range communication. It is also possible to have a nearby restaurant or a gas station sending offers to connected vehicles on their dashboard, or in case of cruise mode cars, speed sign board automatically reduces the speed of car when a message is exchanged between them. Each IoV entity (clustered object, sensors, ECU's) in physical layer will have a cyber entity (one-to-one) created in virtual object layer, which is part of vehicular cloud or cloudlet or fog. MQTT and other IoT topic or content based publish subscribe communication enable publishers (sensors, applications) to publish to certain topics which are subscribed by other sensors or applications, and message broker passes relevant messages to desired subscribers whenever a publisher publishes on these topics. Besides cross entity interactions, in-vehicle communication also occurs, where sensors, ECUs and applications in a connected car exchange messages or interact with a smart device of a passenger sitting inside the car. In-vehicle communication is supported by fog architecture for each car where virtual entities can be created for each ECU, sensor or device. Further, in case of CAN bus communication critical ECUs are separated using gateway which also provides external interface to connected car. This ensures authentication and authorization to over-the-air (OTA) updates and enforces access control policies for in-vehicle communication.

Access control points are needed at physical, virtual object and cloud services layer, where the interaction and data exchange between legitimate and authorized entities is only allowed. At object layer V2V, V2I and other V2X communications using DSRC, WiFi etc. between clustered objects need access controls to ensure BSM confidentiality and prevent malicious activity. Direct access of user using a remote key to unlock a car or through a smart-phone application also needs authorization. It is also possible to store credit card information on vehicle storage or with a cyber entity of the vehicle, which can ease payment process on a toll road, or in a parking garage. In such cases, only authorized applications can access credit card information, which if leaked to

nefarious actors can have huge financial implications. Within object layer, access controls are also needed to ensure authorized communication among sensors or applications and clustered devices, for example in case smart-phone accessing info-tainment systems or plug-in device into car needs security. Access controls are needed when physical objects communicate to their virtual entities in the cloud. For example, an airbag ECU or sensor in the car should only be able to contact its corresponding virtual entity to update its state or push messages via topics. Our concept of IoV ecosystem incorporates virtual objects (for every physical object) which will be important for message and information exchange among heterogenous objects. Virtual entity will be also created for smart devices inside the car that can issue commands to connected vehicle. Therefore, access control is required at virtual object layer also which will control interaction between cyber entities. In-built applications in cars also access on-board sensors for example, tire-pressure monitoring, lane-departure warning system etc., which must be authorized to legitimate applications only. Communication between ECUs also needs authorization using gateway or TCUs. Attribute based access controls can provide fine grained policies and use contextual information to secure data exchange and communication for both physical and virtual object layer. Hence, to secure critical ECUs first level access control restricts external interface and then in-vehicle access control provides second level check.

Connected cars generate lot of data and are referred as ‘data-centers on wheels’. Applications use this data to provide real time information regarding traffic, road safety, weather, or road maintenance. Applications can also diagnose issues of vehicles and offer predictive and precautionary advices to the drivers on road. Such actions through mechanics or users via cloud must be authorized. Further, access controls are required for applications and virtual objects communication, in case any application wants to send a command to a sensor in car. Data generated can be sent to cloud servers for storage and processing. As most of these applications are relevant to geography they can harness the vehicular cloud and use its storage and processing capabilities. Data security is important in the cloud. Proper access controls are required to allow only relevant entities to access and process the data in multi-tenant data lake. Applications and cloud services must be au-

thorized to ensure privacy of user data. The most common platform to analyze big data is Hadoop where several access control models have been proposed including [84–87]. This data can be used by applications inside vehicles or user applications at E-ACO application layer.

4.5.2 Multiple Cloud System

Some smart car applications and use-cases require multiple cloud instances to offer services in vast geographic area or non-time sensitive conditions. For example, assume a vehicle manufacturer has a private cloud where it gathers all data generated from its vehicles, performs analysis for potential problems and offers over the air (OTA) solutions like firmware or software updates. This data can sometimes also reflect problems in the vehicle which needs immediate attention and hence to be sent to a mechanic nearby. Now, the mechanic has its own private cloud and cannot access the vehicle's data which is stored in original equipment manufacturer (OEMs) cloud. In such a scenario, trust has to be established among two clouds so that vehicle's data can be shared between mechanic's cloud and manufacturer's cloud, with the approval of vehicle owner. If a mechanic needs to send messages to sensors in vehicle, cross cloud communication must take place between vehicular cloud (where virtual object of sensor in the car is created) and application in mechanic cloud, which also needs access controls and trust.

Applications like CarSpeak [116] gather data from different sensors not only in the same car but across different cars which may or may not be in the same vehicular cloud. In such cases, the application will access the virtual entity across different vehicular clouds also, which may require trust across different cloud infrastructures. It is also possible to have two vehicular clouds or a vehicular cloud and central cloud exchanging information. For example, suppose a vehicle is approaching the driver's home, and it needs to send a message to the thermostat to turn on the air conditioner. It is possible that the home is in a different cloud, and hence will have its cyber entity in other cloud. In such a scenario cross cloud communication will take place where the application from vehicle will communicate with the virtual object corresponding to the thermostat in the home in other cloud. Since, in this case the home and vehicle belongs to the same owner, we can create

a level of trust between them across clouds and use it to make faster access decision without using policy based controls. In another example, suppose department of motor vehicle (DMV) or local police issues a notice about a stolen car or some nefarious elements in city, a vehicle dashboard will start displaying alert messages. These applications will be running in DMV cloud or cloud owned by police department, which will send messages to the cars running in the city, which also requires multi cloud access scenarios. In such cases, DMV can also have dedicated infrastructure installed around the city or highway which will receive messages sent over cloud and will then pass to nearby vehicles or relevant sensors (through cyber objects, WiFi communication or DSRC) within a defined geographic area.

Henceforth, access controls across single and multiple cloud architectures are needed to ensure secure interaction, data exchange and resource access among physical, virtual objects and applications in Internet of Vehicles ecosystem.

4.6 Dynamic Groups and ABAC for Cloud Assisted Smart Cars

Smart cars location-based services enable notifications and alerts to vehicles. A user must be allowed to set his personal preferences whether he wants to receive advertisements or filter out which ones are acceptable. For instance, a user may not want to receive restaurant notifications but is interested in flash-flood warnings. System wide policy, like a speed warning to all over-speeding vehicles or a policy of who can control speed of autonomous car are needed.

In this section, we present dynamic groups and attribute-based access control (ABAC) model (referred as CV-ABAC$_{\mathrm{G}}$) to ensure authorized data exchange and resource access using secure cloud based communication channels in smart vehicles ecosystem. This model takes into account the user-centric privacy preferences along with system-defined policies to make access decisions. This work proposes a novel concept of groups in context of smart cars, which are dynamically assigned to moving entities like vehicles, based on their current GPS coordinates, direction or other attributes, to ensure relevance of location and time sensitive notification services offered to drivers, provide administrative benefits to manage large numbers of entities and enable attributes

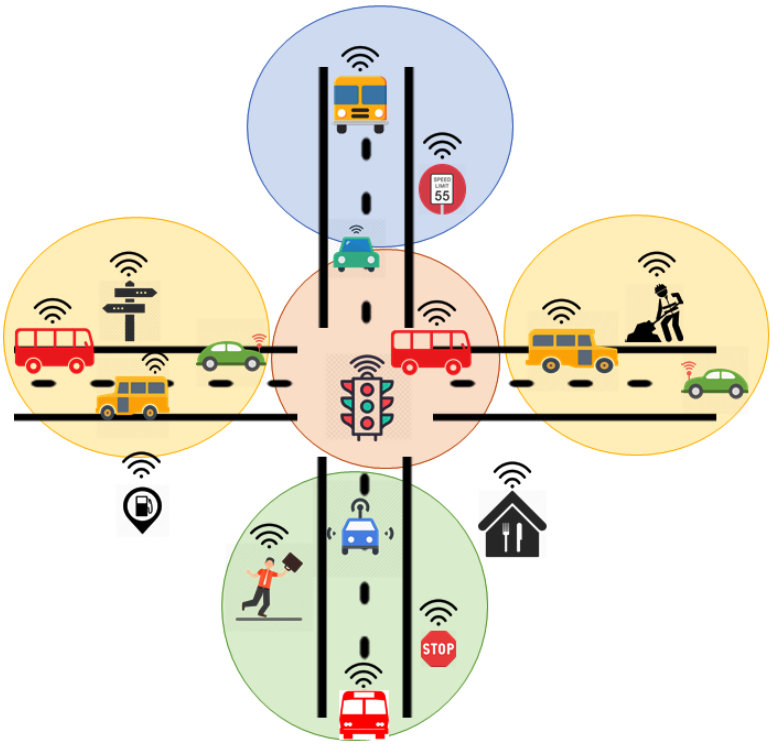


Figure 4.7: Smart City with Location Groups

inheritance for fine-grained authorization. Later in Section 4.8, we present a proof of concept implementation of the proposed model in Amazon Web Services (AWS) IoT platform demonstrating real-world uses cases along with performance metrics.

4.6.1 Relevance of Groups

Most smart cars applications and service requests from drivers are location specific and time sensitive. For example, a driver might want to get warning signals when traveling near a blind spot, in school zone or pedestrians crossing road. Further, notifications sent to drivers are short-lived and mostly pertinent around current GPS coordinates. A gas discount notification from a nearby station, an accident warning two blocks away or ice on the bridge, are some example where alerts are sent to all vehicles in the area. Therefore, we believe that dynamically categorizing connected vehicles into location groups will be helpful for scoping the vehicles to be notified instead of a general broadcast and reduce administrative overheads, since single notification for the group will trigger alerts for all its members. Also, entities present at a location have certain characteristics

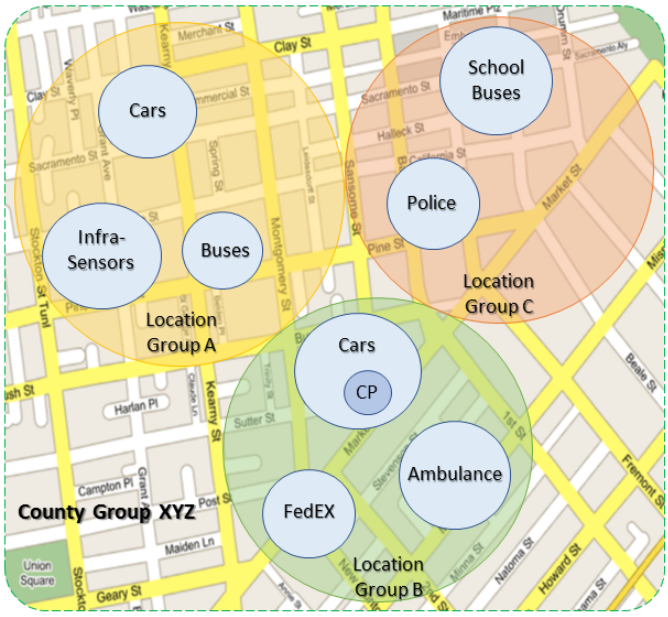


Figure 4.8: Representative Groups Hierarchy

(like stop sign warning, speed limit, deer-threat etc.) in common, which can be inherited by being a group member. Figure 4.7 represents how various smart entities can be separated into different location groups defined by appropriate authorities in a smart city system. These groups are dynamically assigned to connected vehicles based on their attributes, personal preferences, interests or current GPS coordinates as further elaborated in the model and implementation section.

Groups hierarchy can also exist, as shown in Figure 4.8, with sub-groups within a larger parent group so as to reduce the number of vehicles to be notified. For instance, under location group, sub-groups can be created for cars, buses, police vehicles or ambulances, to enable targeted alerts to ambulances or police vehicle sub-groups defined within the location group. Groups can be defined based on services, for example, a group of cars within the car parent group which take part in car-pooling (CP) service or those which want to receive gas station offers. Group hierarchy [88, 153] also enables attributes inheritance from parent to child groups which helps in bulk assignment of attributes to the member entities reducing the overhead of attributes administration.

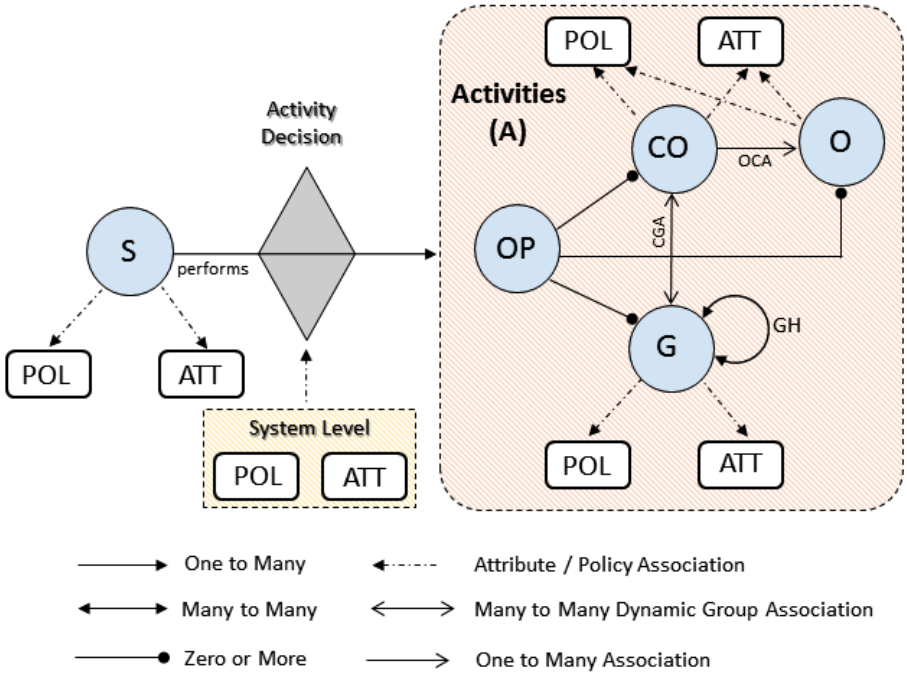


Figure 4.9: A Conceptual CV-ABAC$_{\mathrm{G}}$ Model

4.7 Connected Vehicle ABAC Model with Dynamic Groups

Dynamic communication and data exchange among entities in connected vehicles ecosystem require multi-layer access control policies, which are managed centrally and also driven by individual user preferences. Therefore, an access control model must incorporate all such user and system requirements and offer fine-grained authorization solutions. In this section, we will discuss and formally define our proposed connected vehicle attribute-based access control model with dynamic groups, which we refer as CV-ABAC$_{\mathrm{G}}$.

4.7.1 CV-ABAC$_{\mathrm{G}}$ Model Overview

The conceptual CV-ABAC$_{\mathrm{G}}$ model is shown in Figure 4.9 with formal definitions summarized in Table 4.1. The basic model has following components: Sources (S), Clustered Objects (CO), Objects in clustered objects (O), Groups (G), Operations (OP), Activities (A), Authorization Policies (POL), and Attributes (ATT).

Sources (S): These entities initiate activities (explained below) on various smart objects, groups

and applications in the ecosystem. A source can be a user, an application, administrator, sensor, hand-held device, clustered object (such as a connected car), or a group defined in the system. For instance, in case of flash flood warning, activity source is police or city department triggering an alert to all vehicles in the area. Similarly, mechanic is a source, when he tries to access data remotely from on-board sensor in a connected car.

Clustered Objects (CO): Clustered objects are particularly relevant in case of connected vehicles, traffic lights or smart devices held by humans as they have multiple sensors and actuators. A smart car with on-board sensors, ECUs (like tire pressure, lane departure, or engine control) and applications is a clustered object. These smart entities interact and exchange data among themselves and with others such as requestor source, applications or cloud. An important reason to incorporate clustered objects is to reflect cross-vehicle and intra-vehicle communication. The fact that two smart vehicles can exchange basic safety messages (BSM) shows clustered object communication.

Objects in clustered objects (O): These are individual sensors, ECUs and applications installed in clustered objects. Objects in smart cars include sensors for internal state of the vehicle, e.g., engine diagnostics, emission control, cabin monitoring system, as well as sensors for external environment such as cameras, temperature, rain, etc. Control commands can directly be issued to these objects, and data can be read remotely. Applications (like lane departure warning) on board can also access data from these objects to provide alerts to driver or to a remote service provider.

Groups (G): A group is a logical collection of clustered objects with similar characteristics or requirements. With these groups, subset of COs can be sent relevant notification and also attributes can be assigned to group members. Some groups which can be defined in the smart vehicles ecosystem include location specific groups, service specific groups (like car-pooling, gas station promotions etc.) or vehicle type (a group of cars, buses etc.). Group hierarchy (GH) also exists which enables attributes and policies inheritance from parent to children groups. For simplicity, we require that a vehicle or CO can be direct member of only one group at same hierarchy level. For example, a car can be in either location A or B group and but not both. Such restriction helps in managing attributes inheritance and enhances the usability of our model.

Operations (OP): These are actions which can be performed against clustered objects, individual objects or groups. Examples include: a mechanic performing read, write or control operations on engine ECU, a restaurant triggering notifications to vehicles in location A group. Operations also include administrative actions like creating or updating attributes or policies for COs, objects and groups, which are usually performed by system/security administrators.

Activities (A): Activities encompass both operational and administrative activities which are performed by various sources in the system. An activity can have one or many atomic operations (OP) involved and will need authorization policies, which can be user privacy preferences, system defined or both, to allow or deny an activity. For example, a car pooling notification activity generated by a requestor (source) will be broadcast to all relevant vehicles in the locations nearby using location groups, however individual drivers must also receive or respond to that request based on individual preferences. A driver may not want to car-pool the requestor because of poor rating or because he is not going to the destination the requestor asked for. Therefore, an activity can involve multiple set of policies defined at different levels which must be evaluated, in car-pooling case a policy is set to determine cars to be notified and then driver personal preferences. We have primarily divided these smart car activities into following categories.

- Service Requests: These are activities initiated by entities or users (via on-board car or hand-held applications). For instance, a vehicle break-down initiates a service request to other vehicles around, or a user using a smartphone initiates a car-pooling requests for a destination to the cars which are available for the service.

- Administration: These activities perform administrative operations in system which include changing policies and attributes of entities or determining the group hierarchy. These activities also defines the geographical scope of groups, how user defined privacy preferences are used, or how vehicles are determined to be a member of a group etc.

- Notifications: These are group centric activities where all members are notified for any updates about the group (like speed limit or deer threat notifications in location A) or for

locations-based marketing promotions by parking lots or restaurants.

- Control and Usage: These activities include simple read, write or control operations performed remotely or within a vehicle. Over the air updates issued by manufacturer or turning on car climate control using a smart key are remote activities whereas a passenger accessing infotainment system using smartphone or on-board applications reading camera are local.

Authorization Policies and Attributes: CV-ABAC$_{\mathrm{G}}$ model incorporates individual user privacy controls for different entities by managing authorization policies and entity attributes. A shown in Figure 4.9 policy of sources include personal preferences, whereas attributes reflect characteristics like name, age or gender. Policies can be defined for clustered objects, for instance, a USB can be plugged only by car owner, or which mechanic can access an on-board sensor. Attributes of a car include GPS coordinates, speed, heading direction, and vehicle size. Groups also set policies and attributes for themselves, for example, car pooling group policy of who can be its member. Similarly, system wide policies are also considered, for instance, policy to determine which groups will be sent information when a request comes from a source, or policy to change group hierarchy. Policies also include attributes of entities involved in an activity. A CO can inherit attributes from dynamically assigned groups which will change as the CO leaves old group and adds to new group.

It should be noted that attributes of entities change more often than system wide or individual policies. Attributes are more dynamic in nature which are added or removed with the movement of vehicles or change in surroundings, like GPS coordinates or temperature. Policies once set by administrators or users are more static and only the attributes which comprise the policy change the outcome of a policy but the policy definition remains relatively fixed. For instance, a user policy could state that 'Send restaurant notifications only from Cheesecake factory'. In such case, only attribute name of the restaurant sending the notification will be checked and if it is equal to Cheesecake factory will be able to advertise to that user. Dynamic policies are also possible, for instance, a policy may state that police vans in locations groups A and B are notified in case of emergency, but, in case of a bigger threat this policy can be changed or overwritten with police vans in groups A, B C and D. It should be noted that the CV-ABAC$_{\mathrm{G}}$ model assumes that no policies or

Table 4.1: Formal CV-ABAC$_{\mathrm{G}}$ Model Definitions for Connected Vehicles Ecosystem

Basic Sets and Functions

– S, CO, O, G, OP are finite sets of sources, clustered objects, objects, groups and operations respectively [blue circles in Figure 4.9].
– A is a finite set of activities which can be performed in system.
– ATT is a finite set of attributes associated with S, CO, O, G and system-wide.
– For each attribute att in ATT, Range(att) is a finite set of atomic values.
– attType: ATT = {set, atomic}, defines attributes to be set or atomic valued.
– Each attribute att in ATT maps entities in S, CO, O, G to attribute values. Formally,

$$\text{att} : \text{S} \cup \text{CO} \cup \text{O} \cup \text{G} \cup \{\text{system-wide}\} \rightarrow \begin{cases} \mathrm{Range}(\mathrm{att}) \cup \{\perp\} & \text{if attType(att) = atomic} \\ 2^{\mathrm{Range}(\mathrm{att})} & \text{if attType(att) = set} \end{cases}$$

– POL is a finite set of authorization policies associated with individual S, CO, O, G.
– directG : $\mathrm{CO} \rightarrow \mathrm{G}$, mapping each clustered object to a system group, equivalently CGA $\subseteq \mathrm{CO} \times \mathrm{G}$.
– parentCO : $\mathrm{O} \rightarrow \mathrm{CO}$, mapping each object to a clustered object, equivalently OCA $\subseteq \mathrm{O} \times \mathrm{CO}$.
– $\mathrm{GH} \subseteq \mathrm{G} \times$ G, a partial order relation $\succeq_g$ on G.
Equivalently, parentG : $\mathrm{G} \rightarrow 2^{\mathrm{G}}$, mapping group to a set of parent groups in hierarchy.

Effective Attributes of Groups, Clustered Objects and Objects (Derived Functions)

– For each attribute att in ATT such that attType(att) = set:
- $\mathrm{effG_{att}} : \mathrm{G} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effG_{att}(g_i)} = \mathrm{att(g_i)} \cup \left(\bigcup_{\mathrm{g} \in \{\mathrm{g_j} | \mathrm{g_i} \succeq_g \mathrm{g_j}\}} \mathrm{effG_{att}(g)} \right)$.
- $\mathrm{effCO_{att}} : \mathrm{CO} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effCO_{att}(co)} = \mathrm{att(co)} \cup \mathrm{effG_{att}(directG(co))}$.
- $\mathrm{effO_{att}} : \mathrm{O} \rightarrow 2^{\mathrm{Range(att)}}$, defined as $\mathrm{effO_{att}(o)} = \mathrm{att(o)} \cup \mathrm{effCO_{att}(parentCO(o))}$.

– For each attribute att in ATT such that attType(att) = atomic:
- $\mathrm{effG_{att}} : \mathrm{G} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$,
  defined as $\mathrm{effG_{att}(g_i)} = \begin{cases} \mathrm{att(g_i)} & \text{if } \forall \mathrm{g'} \in \mathrm{parentG(g_i)}.\ \mathrm{effG_{att}(g')} = \perp \\ \mathrm{effG_{att}(g')} & \text{if } \exists\ \mathrm{parentG(g_i)}.\ \mathrm{effG_{att}(parentG(g_i))} \neq \perp \text{ then select} \\ & \text{parent } \mathrm{g'} \text{ with } \mathrm{effG_{att}(g')} \neq \perp \text{ updated most recently.} \end{cases}$
- $\mathrm{effCO_{att}} : \mathrm{CO} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$,
  defined as $\mathrm{effCO_{att}(co)} = \begin{cases} \mathrm{att(co)} & \text{if } \mathrm{effG_{att}(directG(co))} = \perp \\ \mathrm{effG_{att}(directG(co))} & \text{otherwise} \end{cases}$
- $\mathrm{effO_{att}} : \mathrm{O} \rightarrow \mathrm{Range(att)} \cup \{\perp\}$,
  defined as $\mathrm{effO_{att}(o)} = \begin{cases} \mathrm{att(o)} & \text{if } \mathrm{effCO_{att}(parentCO(o))} = \perp \\ \mathrm{effCO_{att}(parentCO(o))} & \text{otherwise} \end{cases}$

Table 4.2: Formal CV-ABAC$_{\mathrm{G}}$ Model Definitions for Connected Vehicles Ecosystem (Continued)

Authorization Functions (Policies)

– Authorization Function: For each op ∈ OP, $\mathrm{Auth_{op}(s : S, ob : CO \cup O \cup G)}$ is a propositional logic formula returning true or false, which is defined using the following policy language:

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\, \mathrm{x} \in \mathrm{set}.\alpha \mid \forall\, \mathrm{x} \in \mathrm{set}.\alpha \mid \mathrm{set} \triangle \mathrm{set} \mid \ \mathrm{atomic} \in \mathrm{set} \mid \mathrm{atomic} \notin \mathrm{set}$
- $\triangle ::= \subset \mid \subseteq \mid \not\subseteq \mid \cap \mid \cup$
- for att ∈ ATT, i ∈ S ∪ CO ∪ O ∪ G ∪ {system-wide}, attType(att) = set : $\mathrm{set} ::= \mathrm{eff_{att}(i)} \mid \mathrm{att(i)}$
- for att ∈ ATT, i ∈ S ∪ CO ∪ O ∪ G ∪ {system-wide}, attType(att) = atomic : $\mathrm{atomic} ::= \mathrm{eff_{att}(i)} \mid \mathrm{att(i)} \mid \mathrm{value}$

Authorization Decision

– A source s ∈ S is allowed to perform an activity a ∈ A, stated as $\mathrm{Authorization(a : A, s : S)}$, if the required policies needed to allow the activity are included and evaluated to make final decision. These multi-layer policies must be evaluated for individual operations ($\mathrm{op_i}$ ∈ OP) to be performed by source s ∈ S on relevant objects ($\mathrm{x_i}$ ∈ CO ∪ O ∪ G). Formally,
$\mathrm{Authorization(a : A, s : S)} \Rightarrow$
$\mathrm{Auth_{op_1}(s : S, x_1)}$, $\mathrm{Auth_{op_2}(s : S, x_2)}$, ... , $\mathrm{Auth_{op_n}(s : S, x_3)}$

attributes are changed during an activity evaluation session.

Some activities will need multi-level policy evaluation and may include user privacy preferences. For instance, a user must be allowed to decide if he wants to share data from car sensors or whether wants to get marketing advertisements. Each activity will evaluate required system and user defined ecurity policies to make final decision.

4.7.2 Formal Definitions

As shown in Table 4.1, sources, clustered objects, objects and groups can be directly assigned values from the set of atomic values (denoted by Range(att)) for attribute att in set ATT. Each attribute can be set or atomic valued, determined by attType function. Based on attribute type, entities can be assigned a single value including null ($\perp$) for an atomic attribute, or multiple values for set-valued attribute from the attribute range. POL is the set of authorization policies in the system which will be defined below.

Clustered objects can be members of different groups, based on preferences and requirements. For example, a car is assigned to a location group based on its GPS coordinates. In our model,

we assume that a clustered object can be directly assigned to only one group at same hierarchy level (specified by directG function). As we will discuss later that since groups inherit attributes from parent groups, assigning a clustered object to one parent group is sufficient to realize attributes inheritance. Smart cars have sensors and applications installed in them, which can also be accessed by different sources. Therefore, parentCO function determines the clustered object to which an object belongs, which is a one to many mapping i.e an object can only belong to one CO while a CO can have multiple objects. Further, group hierarchy GH (shown as self loop on G), is defined using a partial order relation on G and denoted by $\succeq_g$, where $g_1 \succeq_g g_2$ signifies g_1 is child group of g_2 and g_1 inherits all the attributes of g_2. Function parentG computes the set of parent groups in the hierarchy for a child group.

The benefit to introduce groups is ease of administration where multiple attributes can be assigned or removed from member clustered objects with single administrative operation. Group hierarchy enables attributes inheritance from parent to child groups. Therefore, in case of set valued attributes, the effective attribute att of a group $\mathrm{g_i}$ (denoted by $\mathrm{effG_{att}(g_i)}$) is the union of directly assigned values for attribute att and the effective values for att for all its parent groups in group hierarchy. This definition is well formed since $\succeq_g$ is a partial order. For a maximal group $\mathrm{g_j}$ in this ordering, we have $\mathrm{effG_{att}(g_j)}$ = att($\mathrm{g_j}$), giving base cases for this recursive definition. The effective attribute values of clustered object for attribute att (stated as $\mathrm{effCO_{att}}$) will then be the directly assigned values for att and the effective attribute values of att for the group to which CO is directly assigned (by directG). Similarly, in addition to direct attributes, sensors in car can inherit attributes from the car itself (eg. make, model, location), $\mathrm{effO_{att}}$ calculates these effective attributes of objects. For set valued attributes, union operation will be sufficient which is not true for atomic attributes. In case of groups, the most recently updated non-null attribute values in parent groups will overwrite the values of child group as defined in Table 4.1. For example, if the most recent value updated in one of the parent groups for Deer_Threat attribute is 'ON', this value will trickle to the child group. It should be noted that overwriting with the most recently updated value in groups is one of the many approaches to inherit atomic attributes, but for the dynamic nature

of smart cars ecosystem, we believe this is most appropriate. Clustered object inherits non-null atomic value from its direct parent group as stated by $\mathrm{effCO_{att}(co)}$ = $\mathrm{effG_{att}(directG(co))}$. In case of objects, parent clustered object will overwrite non-null atomic attributes. For atomic attributes, if the parent(s) has null value for an attribute, the entity (group, clustered object or object) will retain its directly assigned value without any overwrite.

Authorization functions are defined for each operation op ∈ OP, which are policies defined in the system. POL is the set of all authorization functions, $\mathrm{Auth_{op}(s : S, ob : CO \cup O \cup G)}$, which specify the conditions under which source s ∈ S can execute operation op ∈ OP on object ob ∈ CO ∪ O ∪ G. Such policies include privacy preferences set by users for individual clustered object, objects and groups or can be system wide by security administrators. The conditions can be specified as propositional logic formula using policy language defined in Table 4.2. Multiple policies must be satisfied before an activity is allowed to perform. Authorization function, $\mathrm{Authorization(a : A, s : S)}$, where an activity a ∈ A is allowed by source s ∈ S, specifies the system level, user privacy policies or other relevant policies returning true for an activity to succeed.

CV-ABAC$_{\mathrm{G}}$ is an attribute-based access control model which satisfies fine-grained authorization needs of dynamic, location oriented and real-time interaction in smart cars ecosystem. The model supports personalized privacy controls by utilizing individual user policies and attributes, along with dynamic groups assignment. CV-ABAC$_{\mathrm{G}}$ assumes that the information and attributes shared by source and object entities are trusted, for instance, location coordinates sent by a car are correct, and uses this shared information to make access decision. How to ensure that the information is from a trusted source or is correct is out of the scope of this work.

4.8 Enforcement in Amazon Web Services

In this section, we present a proof of concept demonstration of CV-ABAC$_{\mathrm{G}}$ model by enforcing a use case of smart cars using AWS IoT service [26]. The implementation will demonstrate how dynamic groups assignment and multi-layer authorization policies required in connected vehicle ecosystem can be realized in AWS. We have used simulations to reflect real connected smart ve-

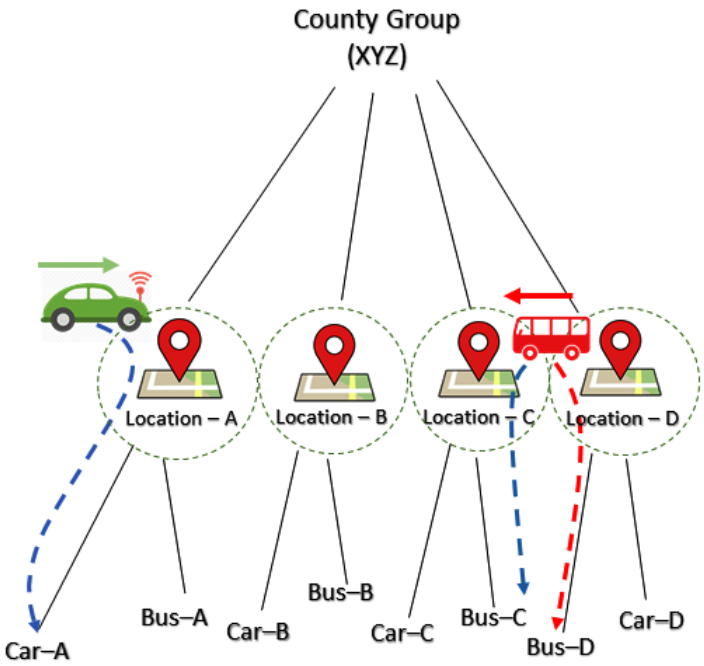


Figure 4.10: Groups Hierarchy in AWS

hicles, however, it does not undermine the plausibility, use and advantage of our proposed model as further elaborated in following discussion. It should be noted that no long term vehicle data including real-time GPS coordinates are collected in central cloud, which mitigates user privacy concerns and encourages wide adoption of the model.

4.8.1 Description of Use Cases

Location based alerts and notifications are important in smart cars applications and motivate our use cases. We will build upon our defined group hierarchy in AWS shown in Figure 4.10. Our implementation will enforce access controls in following use cases.

Deer Threat Notification - Smart infrastructure in the city can sense the surrounding environment and notify group(s) regarding the change. In this use case, a motion sensor senses deers in the area and changes Deer_Threat attribute of location group to ON which in-turn sends alerts to all member vehicles in that location. Similar, implementation can be done in case of accident notification, speed limit warning or location based marketing.

Car-Pooling - A traveller needs a ride to Location-A. Using a mobile application, he sends car-

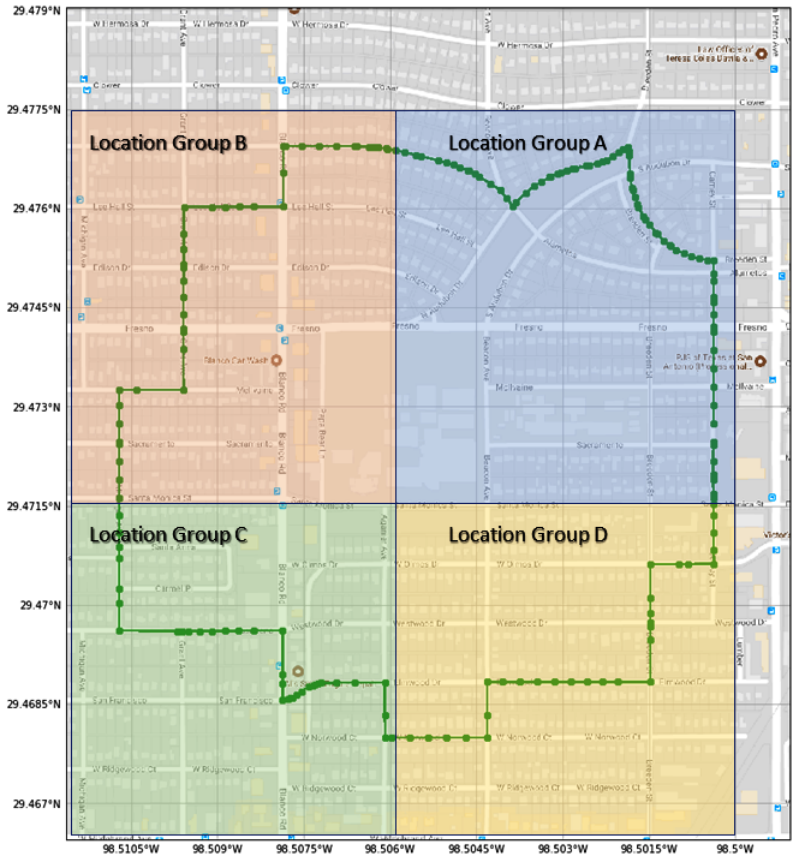


Figure 4.11: Vehicle GPS Coordinates and Groups Demarcation

pooling requests to vehicles in his vicinity which are heading to the destination location asked by the traveller. The request is received by AWS cloud, which computes location and appropriate groups based on the coordinates of the requester, to publish notifications to nearby cars. All the members of group Car-A, B, C or D can get the request, but some cars may not want to be part of car-pooling, or do not want some requestors to join them because of ratings. User policies will be also checked before a driver is notified of likely car-pool customer.

4.8.2 Prototype Implementation

AWS implementation of our model in these use-cases involves two phases: administrative phase and operational phase. Administrative part involves creation of groups hierarchy, dynamic assignment of moving vehicles to different location and sub-groups, attributes inheritance from parent to child groups and to group members, and attributes modification of entities. Operational part covers

```
('Received new coordinates from:', 'Vehicle-1')
Sun May 27 02:56:30 2018
  Location A
     Car-A : [u'Vehicle-1', u'Vehicle-2']
     Bus-A : []
  Location B
     Car-B : []
     Bus-B : [u'Vehicle-6']
  Location C
     Car-C : [u'Vehicle-3', u'Vehicle-4']
     Bus-C : []
  Location D
     Car-D : []
     Bus-D : [u'Vehicle-5']
```

Figure 4.12: Dynamic Groups and Vehicles in AWS

how groups are used to scope down the number of vehicles who receive messages or notifications from different sources. Both these phases involve multi-layer access control polices. We created an ABAC policy decision (PDP) and enforcement point (PEP) [97], and implemented our external policy evaluation engine which is hooked with AWS to enable attribute-based authorization.

Administrative Phase: We created a group hierarchy in AWS as shown in Figure 4.10. In this hierarchy, County-XYZ is divided into four disjoint Location-A, B, C and D groups, with each having Car and Bus subgroups for vehicle type car or bus. We created 10 vehicles and simulated their movement using a python script which publishes MQTT message to shadows of these vehicles with current GPS coordinates (generated using Google API [33]) iterated over green dots shown in Figure 4.11. The area was demarcated into four locations and a moving vehicle belongs to a subgroup in one of these groups. Assuming current location of Vehicle-1 as Location-D, and it publishes MQTT message with payload:

```
{"state": {"reported": {"Latitude": "29.4769353",
                        "Longitude":"-98.5018237"}}}
```

to AWS topic: `$aws/things/Vehicle-1/shadow/update`, its new location changes to Location-A and since we defined the vehicle type as car, it is assigned to Car-A group under

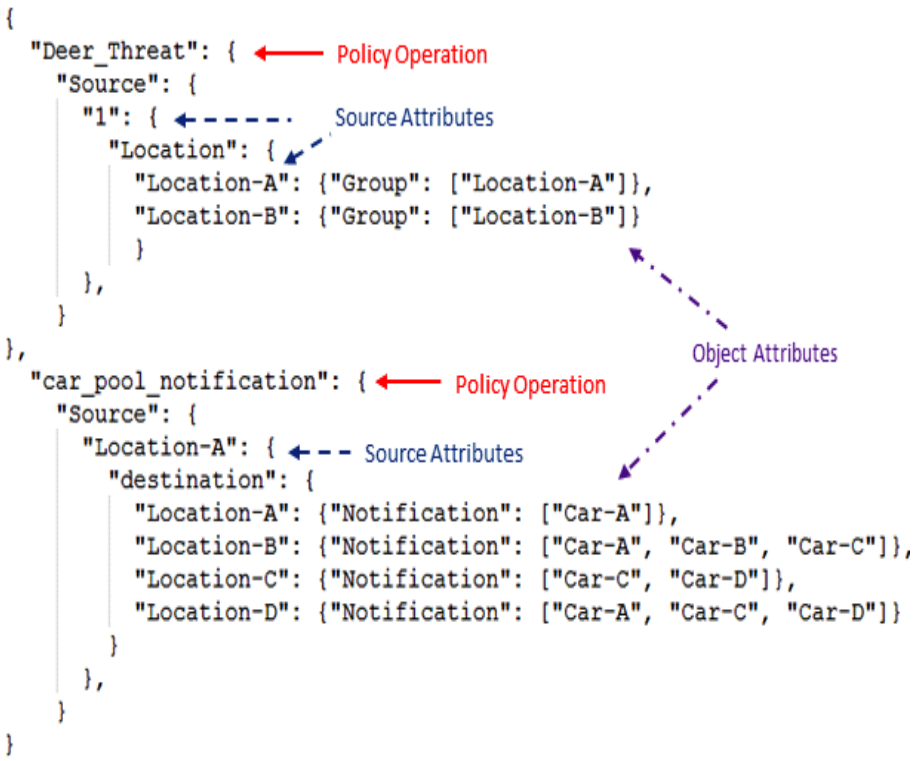


Figure 4.13: Attribute Based Policies in AWS

Location-A as shown by snippet in Figure 4.12. Both attributes, vehicle type and current coordinates of vehicle, are used to dynamically assign groups, which is important in moving smart vehicles. These functionalities are implemented as a stand alone service (can be enforced as a Lambda service [27] function) using Boto [28] which is the AWS SDK for Python. Further, in case of deer threat notification use-case, we simulated a location-sensor which senses deers in the area and updates the attribute 'Deer_Threat' of location group to 'ON' or 'OFF', which is then notified to all members of location and its subgroups. We defined an attribute-based policy to control which sensors can change the 'Deer_Threat' attribute of location groups. As shown in Figure 4.13, our policy for Deer_Threat operation checks that a motion sensor with ID = '1' and current groups of Location-A can update the attribute Deer_Threat for group Location-A, and if sensor is relocated to Location-B it can update attribute for Location-B group only. This policy ensures that the sensor must be in that location group for which it is updating Deer_Threat attribute.

The complete sequence of events performed in AWS along with our stand-alone service for the administrative phase is shown in Figure 4.14. A moving vehicle updates its coordinates to

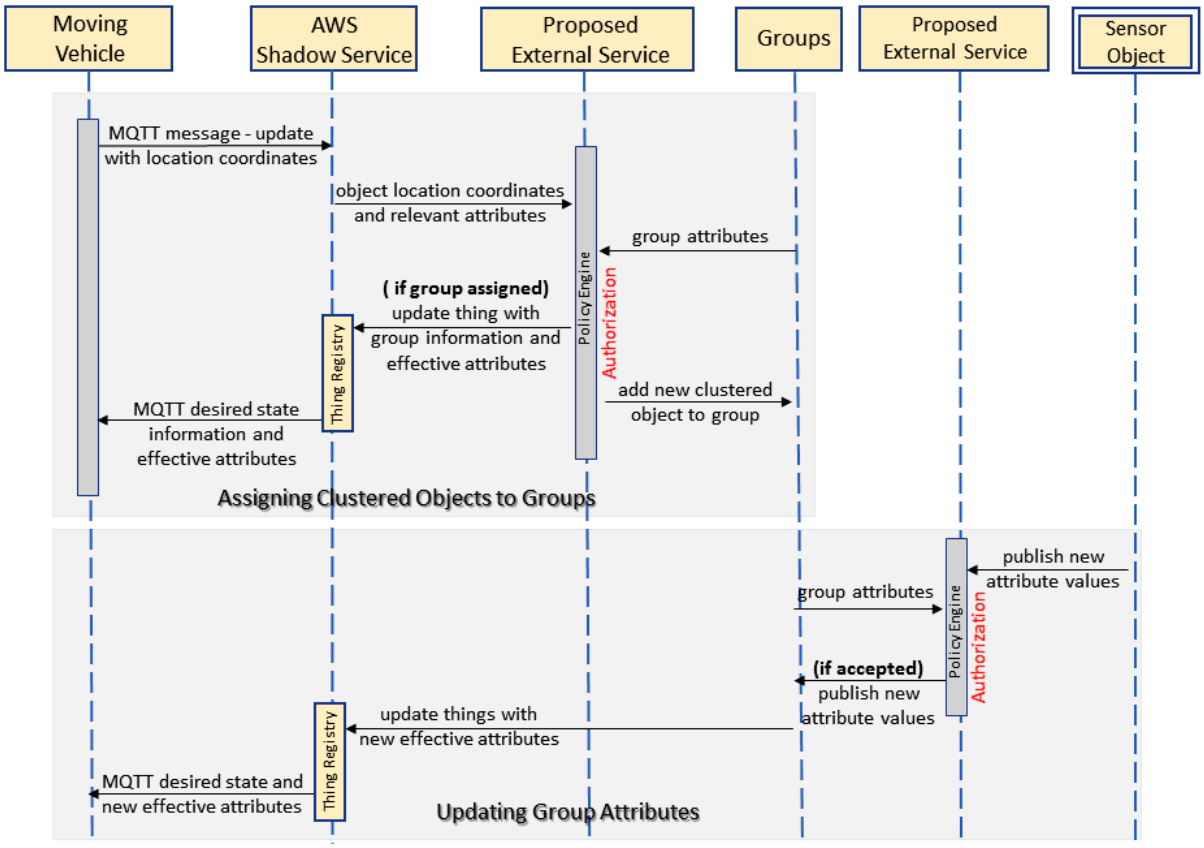


Figure 4.14: Sequence Diagram for Dynamic Groups and Attributes Assignment in AWS

AWS shadow service, which along with attributes of vehicles and location groups determines if the vehicle can be member of the group using our external enforcement service. If authorization policy allows vehicle to be a member of group, the vehicle and group is notified and vehicle inherits all attributes of its newly assigned group. Similarly, if attribute 'Deer_Threat' of group is allowed (by authorization policy) to be changed by the location sensor, the new values are propagated to all its members. We implemented attribute inheritance from parent to child groups through our service using `update_thing_group` and `update_thing` methods. In our use-case attributes inheritance exist from Location-A to all both subgroups Car-A and Bus-A, and to vehicles in Car-A and Bus-A. Therefore, when attribute 'Deer_Threat' is set to ON in group Location-A, its new attributes using Boto `describe_thing_group` command are:

```
{'Center-Latitude': '29.4745', 'Center-Longitude':
'-98.503','Deer_Threat': 'ON'}
```

This inherits the attributes to Car-A child group whose effective attributes will now be:

```
{'Center-Latitude': '29.4745', 'Center-Longitude':
'-98.503','Deer_Threat': 'ON', 'Location': 'A'}
```

As shown in Figure 4.12, both Vehicle-1 and Vehicle-2 as member of group Car-A under Location-A parent group, therefore the effective attributes of Vehicle-2 are:

```
{'Center-Latitude': '29.4745', 'Center-Longitude':
'-98.503','Deer_Threat': 'ON', 'Location': 'A',
'Type': 'Car', 'VIN': '9246572903752', 'thingName':
'Vehicle-2'}
```

Operational Phase: In this phase, attribute-based policies are used to restrict service and notification activities which may require single or multi-level policies along with user preferences. In car-pooling use case, we defined policies to restrict notifications to only a subset of relevant vehicles in specific locations. We simulated requestor in AWS needing car-pool. It has attribute 'destination' with value in Location-A, B, C or D. Requestor sends source and destination location as MQTT message to AWS topic `$aws/things/Requestor/shadow/update` which based on these attributes determine subgroups to which service requests is sent.

```
{"state": {"reported": {"policy":
 "car_pool_notification", "source": "Location-A",
 "destination": "Location-B"}}}
```

The policy for `car_pool_notification` operation (shown in Figure 4.13) suggests that if current location of source requestor is 'Location-A' and destination location is somewhere in 'Location-A' then all members of sub-group Car-A should be notified. Similarly, if the destination attribute is Location-B, then all members of Car-A, Car-B and Car-C needs notification. In our

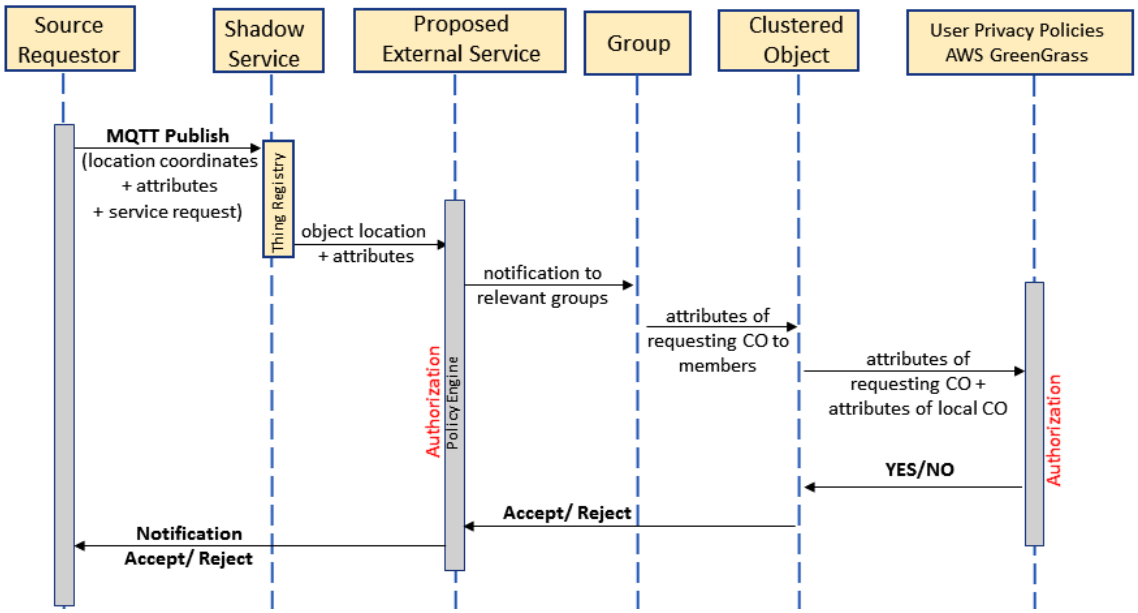


Figure 4.15: Sequence Diagram for Attributes Based Authorization in AWS

use-case, all members of these sub-groups are notified. The policy restricts the number of vehicles which will be requested as compared to all vehicles getting irrelevant notification (as they are far from the requestor or are not vehicle type car) and illustrates the importance of location-centric smart car ecosystem. Similarly, location-based marketing can be restricted and policies can be defined to control such notifications.

User privacy policies take into effect once the subset of vehicles is calculated. These policies encapsulate user preferences, for instance, in car pooling a particular driver is not going to the destination requested by the requestor in his request, therefore notification will not be displayed on his car dashboard. These local policies are implemented using AWS Greengrass [25] which allows to run local lambda functions on the device (in our case a connected vehicle) to enable edge computing facility, an important requirement in real-time smart car applications and enforce privacy policies. Once accepted by drivers, a SNS (AWS Simple Notification Service) [29] message can be triggered for requestor from accepting vehicles along with name and vehicle number. The sequence of events for car-pooling activity and multi-layer authorization policies together with user personal preferences is shown in Figure 4.15.

Our proposed external service to implement ABAC policy decision and evaluation helps

Number of Requests	Policy Enforcer Execution Time (in ms)
10	0.0501
20	0.1011
30	0.1264
40	0.1630
50	0.1999

	Cars Notified	
nth Request	With ABAC Policy	Without Policy
41st	2	5
42nd	3	5
43rd	5	5
44th	3	5
45th	2	5
46th	3	5

Figure 4.16: Policy Enforcement Time and Scoping

achieve fine grained authorization needed in smart cars ecosystem. The implementation also demonstrates dynamic groups assignment based on mobile vehicle GPS coordinates and attributes along with groups based attributes inheritance which offer administrative benefits in enforcing an ABAC model. In this entire implementation, no persistent data from moving vehicles is collected or stored by the central authority hosted cloud which reaffirms its privacy preserving benefits.

4.8.3 Performance Evaluation

We evaluated the performance of our proposed CV-ABAC$_{\mathrm{G}}$ model in AWS and provide different metrics when no policies were used against our implemented ABAC policies for the car-pooling notification use-case. As shown in Figure 4.16, our external policy evaluation engine has average time (in milliseconds) to decide on car-pooling service requests and provide the subset of cars which are notified. This scoping ensures the service relevance as without a policy all 5 vehicles were sent car-pool request (even when one was 20 miles away from the requestor), whereas with attribute based policies only nearby cars are notified. The performance graph shown in Figure 4.17 compares no policy execution time (red line) against implemented ABAC policy (blue line). Since, in our experiments the policy (shown in Figure 4.13) evaluated for each access requests is the same, we get a linear graph as the number of access requests increase the number of times the policy is evaluated and so is its total evaluation time. Some variation in red line is because of the network latency time to access AWS cloud, although this can change based on the communication technologies used by vehicles including 3G, LTE, cellular or DSRC [31]. Clearly, this external

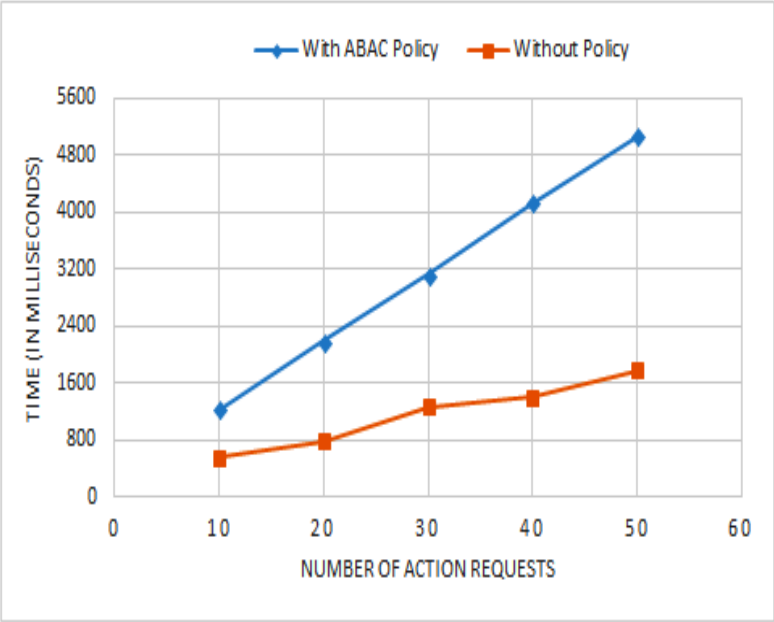


Figure 4.17: Performance Evaluation

policy engine does have some impact on the performance against no policy when used with number of vehicles. However, we believe when used in city wide scenario this time will be overshadowed by the notification time to all vehicles against a subset of vehicles provided by the engine. Our proposed model is focused to ensure relevance of service and data exchange to nearby drivers on road which is well achieved even with a little tradeoff.

It should be noted that the sole purpose of this proof of concept is to showcase the practical viability and use of security policies in context of smart cars, without the need to capture large set of data points from real world traffic scenarios spread across wide geographic area and sizable on-road moving vehicles. Such scaled setting will only stress the entire system without reflecting any change in policy evaluation.

CHAPTER 5: BIG DATA SECURITY IN HADOOP ECOSYSTEM

In this chapter, we discuss authorization requirements in the widely used multi-tenant Big Data processing framework, known as Hadoop. Hadoop ecosystem provides a highly scalable, fault-tolerant and cost-effective platform for storing and analyzing variety of data formats. Apache Ranger [7] and Apache Sentry [8] are two predominant frameworks used to provide authorization capabilities in Hadoop. We first formalize the current access control model for Hadoop Ecosystem and propose two extensions including Object-Tagged RBAC Model and fine grained ABAC model. Significant portions of this chapter have been published at the following venues [85, 87]:

- Maanak Gupta, Farhan Patwa and Ravi Sandhu, Object-Tagged RBAC Model for the Hadoop Ecosystem. In Proceedings of the 31st Annual IFIP WG 11.3 Working Conference on Data and Applications Security and Privacy (DBSec 2017), July 19-21, 2017, Philadelphia, Pennsylvania, USA, pages 63-81.

- Maanak Gupta, Farhan Patwa and Ravi Sandhu, An Attribute-Based Access Control Model for Secure Big Data Processing in Hadoop Ecosystem. In Proceedings of the 3rd ACM Workshop on Attribute-Based Access Control (ABAC 2018), March 19-21, 2018, Tempe, Arizona, USA, pages 13-24.

5.1 Introduction and Motivation

Over the last few years, enterprises have started harvesting data from 'anything' to discover business and customer needs. It is estimated that 163 zettabytes of data will be generated annually by year 2025 as quoted by IDC [11]. Such massive and varied collections of data, referred to as Big Data, are considered 21st century gold for data miners. Enterprises gain useful insights from analysis to offer targeted marketing, fraud detection, accident forecasting, traffic patterns and even strong love matching. With volume, variety and velocity of data burgeoning, massive storage and compute clusters are required for analysis. Data lake formed by the amalgamation of data from these sources requires powerful, scalable and resilient, storage and processing platforms to reveal

the true value hidden inside this data mine.

Apache Hadoop [3] has established itself as an important open-source framework for cost-efficient, distributed storage and computing of data in timely fashion. The platform offers resilient infrastructure for sophisticated analytical and pattern recognition techniques for multi-structured data. Hadoop ecosystem includes several open-source and commercial tools (Apache Hive [5], Apache Storm [9], Apache HBase [4], Apache Ambari [2] etc.) built to leverage the full capabilities of Hadoop framework. These tools along with Apache Hadoop 2.x core modules (Hadoop Common, Hadoop Distributed File System (HDFS), YARN and MapReduce) empower users to harness the potential of data assets.

As Hadoop is widely used in government and private sector, its security has been a major concern and widely studied. Multi-tenant Data Lake offered by Hadoop, stores and processes sensitive information from several critical sources, such as banking and intelligence agencies, which should only be accessed by legitimate users and applications. Threats—including denial of resources, malicious user killing YARN applications, masquerading Hadoop services like NameNode, DataNode etc.—can have serious ramifications on confidentiality and integrity of data and ecosystem resources. In year 2017 alone several instances of data breaches [21] were brought to the notice of the world [11] which amplifies and emphasises the need for better cyber security and privacy mechanisms. The distributed nature and platform scale of Hadoop makes it more difficult to protect the infrastructure assets.

Apache Ranger [7] and Apache Sentry [8] are two important software systems used to provide fine-grained access across several Hadoop ecosystem components. In this chapter we present the multi-layer access control model for Hadoop ecosystem (referred as HeAC), formalizing the authorization model in Apache Ranger (release 0.6) and Sentry (release 1.7.0) in addition to access controls capabilities in core Hadoop 2.x. We further propose an Object-Tagged Role Based Access Control (OT-RBAC) model which leverages the merits of RBAC and provides a novel approach of adding object attribute values (called object tags) in RBAC model. We also outline extensions to OT-RBAC to incorporate NIST proposed strategies [114] for adding attributes in RBAC. Im-

plementation and evaluation of OT-RBAC using Apache Ranger is also discussed in this chapter. Further, we present a formalized attribute based access control model for Hadoop ecosystem, referred to as HeABAC. We also present an implementation approach for HeABAC using Apache Ranger along with comprehensive real-world use cases to reflect the application and enforcement of the proposed HeABAC model in Hadoop ecosystem.

5.2 Multi-layer Authorization

The most critical assets required to be secured in Hadoop involve services, data and service objects, applications and cluster infrastructure resources. In this section, we discuss the multi-layer authorization capabilities provided in Hadoop ecosystem in line with Apache Hadoop 2.x, along with access control features offered by Apache Ranger, Apache Sentry and Apache Knox [6].

5.2.1 Hadoop Services Access

The first layer of defense is provided by service level authorization which checks if a user or application is allowed to access the Hadoop ecosystem services and Hadoop core daemons. This check is done before data and service objects permissions are evaluated, thereby preventing unauthorized access early in the access request lifecycle. ACLs (Access Control Lists) are specified with users and groups to restrict access to services. For example, ACL `security.job.client.protocol.acl` is checked to allow a user to communicate with YARN ResourceManager for job submission or application status inquiry. This layer also restricts cross-service communication to prevent malicious processes interaction with Hadoop daemon services (NameNode, ResourceManager etc.). Another ACL `security.datanode.protocol.acl` is checked for interaction between DataNodes and NameNode for heartbeat or task updates. A user making API requests to individual ecosystem services like Apache Hive, HDFS, Apache Storm etc., is restricted by implementing single gateway (e.g Apache Knox [6]) access point—which enforces policies to allow or deny users to access ecosystem services before operating on underlying objects.

5.2.2 Data and Service Objects Access

Hadoop Distributed File System (HDFS) enforces POSIX style model and ACLs for setting permissions on files and directories holding data. Multiple other ecosystem services require different objects to be secured. For example, Apache Hive requires table and columns, whereas Apache Kafka secures topic objects from unauthorized operations by users. Some services like Apache Hive or Apache HBase also have native access control capabilities to secure different data objects. Security frameworks like Apache Ranger or Sentry provide plugins for individual ecosystem services, where centralized policies are set for different data and service objects. In Apache Ranger, authorization policies can also be formulated on Tags, which are attribute values associated with objects. For example, a tag PII can be associated with table SSN and a policy created for tag PII. Such tag-based policy will then control access to table SSN. Tags allow controlling access to resources along several services without need to create separate policies for individual services. It should be noted that data access allowed at one service may be restricted by permissions at underlying HDFS, thereby requiring user to have multiple object permissions at different services.

5.2.3 Application and Cluster Resources Access

Multi-tenant Hadoop cluster requires sharing of finite resources among several users, controlled by Apache YARN capacity (or fair) scheduler queues in Hadoop 2.x. Queue level authorization enables designated users to submit or administer applications in different queues. This restricts user from submitting applications in cluster and prevents rogue users from deleting or modifying other user applications. Further, cluster resources are not consumed by certain applications requiring more resources as queues have limited resources allocated. It should be noted that application owner and queue administrator can always kill or modify jobs in queue. These queues support hierarchical structure where permissions to parent queues descend to all child queues. Hadoop implements these authorization configurations using ACLs. Configuration file can also be associated with applications to specify users who can modify or kill an application. Access to cluster nodes can be restricted by assigning node labels. Each queue can be associated with node labels to

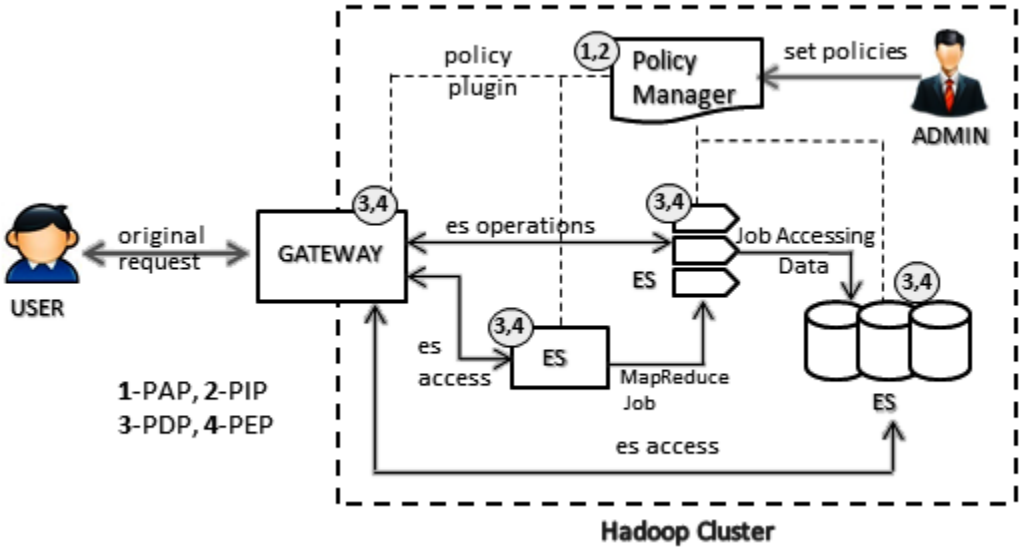


Figure 5.1: Example Hadoop Ecosystem Authorization Architecture

restrict nodes where applications submitted to queues can run.

Figure 5.1 illustrates multi-layer authorization architecture provided in Hadoop ecosystem. An authenticated user passes through several access control mechanisms to operate on objects and services in Hadoop cluster. Gateway (such as Apache Knox) offers single access point to all REST APIs and provides first layer of access control to check if services inside the cluster are allowed access by outside users. Once user is approved through the gateway policy plugin, ecosystem services apply policies cached from central policy manager to validate requests of user. User trying to access objects (files, tables etc.) in ecosystem services like HDFS or Apache Hive (shown as ES in Figure 5.1) is checked by policy plugins attached to the services to enforce access decisions. A user wanting to submit an application or to get submitted application status should be allowed through gateway policies to communicate with YARN ResourceManager. Apache YARN queue permissions are then checked and enforced by plugin to know if a user is allowed to submit or administer application in queues. Cross-services access (between Hadoop daemons) for information passing or task status update is mainly enforced using core Hadoop service ACLs.

As shown in Figure 5.1, security frameworks like Apache Ranger provides centralized Policy adminstration (1-PAP) and information point (2-PIP). Enforcement and decision points (3-PEP,

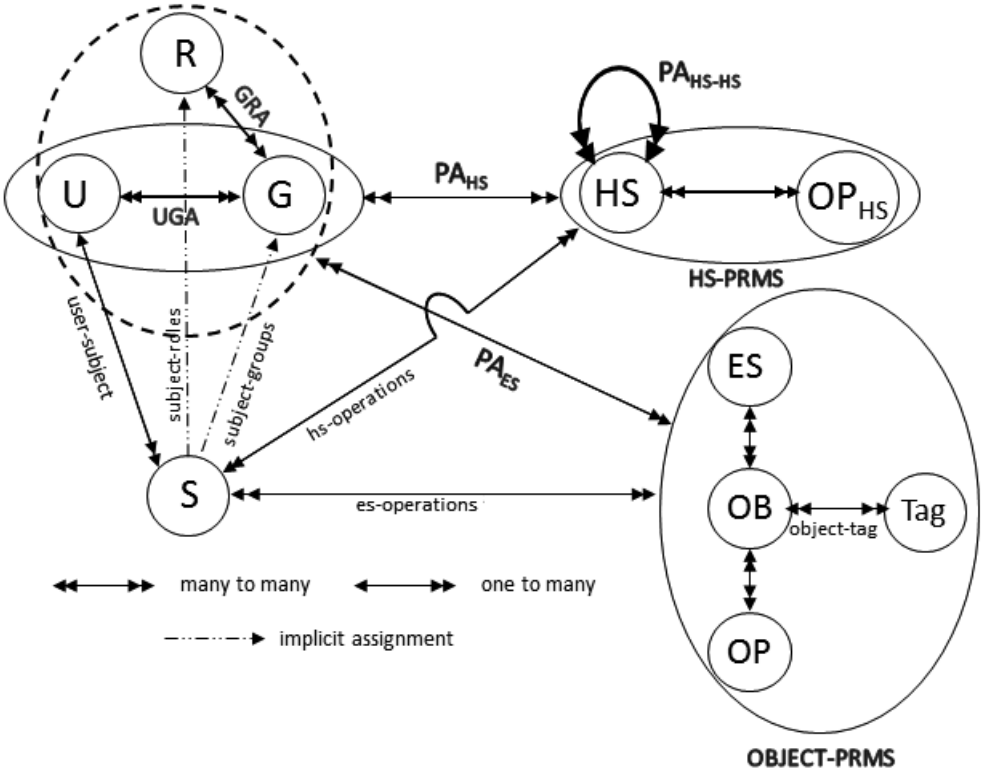


Figure 5.2: A Conceptual Model of HeAC

4-PDP) are plugins attached to each service which cache policies periodically from central server and enforce access decisions.

5.3 Hadoop Ecosystem Access Control Model

In this section we present the formal multi-layer access control model (HeAC) for Hadoop ecosystem based on Apache Hadoop 2.x. The model also covers access capabilities provided by two predominant Apache projects, Ranger (release 0.6) and Sentry (release 1.7.0). Apache Ranger supports permissions through users and groups, while Sentry assigns permissions to roles which are assigned to groups and via groups to member users. We will now discuss the formal definitions of HeAC model as shown in Figure 5.2 and specified in Table 5.1.

The basic components of HeAC include: Users (U), Groups (G), Roles (R), Subjects (S), Hadoop Services (HS), Operations ($\mathrm{OP_{HS}}$) on Hadoop Services, Ecosystem Services (ES), Data and Service Objects (OB) belonging to Ecosystem Services, Operations (OP) on objects, and Ob-

ject Tags (Tag).

- **Users, Groups, Roles and Subjects:** A user is a human who interacts with the system to access services and objects inside the Hadoop ecosystem. A group is a collection of users in the system with similar organizational requirements. A role is a collection of permissions which can be assigned to different entities in the system. Permissions are assigned to users, groups or roles. In the current model roles can only be assigned to groups, thereby giving permissions to member users of groups indirectly. A subject is an application running on behalf of the user to perform operations in the Hadoop ecosystem. In HeAC model subjects always run with full permissions of the creator user.

- **Hadoop Services:** These services are background daemon processes, like HDFS NameNode, DataNode, YARN ResourceManager, ApplicationMaster etc., which provide core functionalities in Hadoop 2.x framework. User access these services to submit applications, data block recovery or application status updates. Besides interaction with end user, these daemon services also communicate with each other for resource monitoring or task updates. It should be noted that these services do not have objects associated with them.

- **Operations on Hadoop Services:** These are actions allowed on Hadoop services. In most cases, the general action allowed is to access a service. For example, ACL `security.client.protocol.acl` is used to determine which user is allowed to access HDFS NameNode service. These ACLs are part of Hadoop native access control capabilities (referred as service level authorization).

- **Ecosystem Services:** Data and objects inside the Hadoop ecosystem are accessed through different platforms which we consider as Ecosystem Services. Example of such services include Apache HDFS, Apache Hive, Apache HBase, Apache Storm, Apache Kafka etc. These services can either have data objects (tables, columns) or other type of resources (queues, topics) which they support. Access to the ecosystem services is first required before

operation on supported objects. We consider Data Services as one instance of Ecosystem Services.

- **Data and Service Objects:** Ecosystem services manage different types of resources (objects) inside the cluster. For example, Apache HDFS supports files and directories, while Apache HBase has data objects like column-family, cells etc. YARN manages queue objects and Apache Solr has collections. These are resources which are protected from unauthorized operations from users.

- **Operations on objects:** Multiple data and service objects support different operations to perform actions on them. Apache Hive has select, create, drop, alter for tables and columns while Apache HBase data objects (column family, column) support read, write, create etc. YARN queues have operations to submit applications or administer the queue.

- **Object Tags:** Objects inside ecosystem can be assigned attributes based on sensitivity, content or expiration date. Such classification is done using attribute values called Tags. An object can have multiple tags associated with it and vice versa. For example, PII tag can be attached to sensitive data table SSN.

As shown in Table 5.1, a user can be assigned to multiple groups defined by $\mathrm{directUG}$ function. Groups are also assigned to multiple roles as reflected by function $\mathrm{directGR}$. Relation $\mathrm{object\text{-}tag}$ denotes a many-to-many relation between objects and associated attribute values called tags. Hadoop ecosystem has two different sets of permissions to perform actions on services and objects. $\mathrm{OBJECT\text{-}PRMS}$ is the set of data and service object permissions which is power set of the cross product of ecosystem services (ES), objects (OB) or object tags (Tag), and operations (OP). Here permissions can be set either on object or object tags, and policies can allow or deny operations on the object based on its associated tags or the object itself. $\mathrm{OBJECT\text{-}PRMS}$ also include ecosystem service as part of permission thereby taking into account the requirement of service access before object operations. Another set of permissions called Hadoop service permissions ($\mathrm{HS\text{-}PRMS}$) is the power set of the cross product of HS and $\mathrm{OP_{HS}}$. These are required for

Table 5.1: Hadoop Ecosystem Access Control (HeAC) Model Definitions

Basic Sets and Functions
– U, G, R, S (finite set of users, groups, roles and subjects respectively)
– HS, $\mathrm{OP_{HS}}$ (finite set of Hadoop services and operations respectively)
– ES, OB (finite set of ecosystem services and objects respectively)
– OP, Tag (finite set of object operations and object tags respectively)
– $\mathrm{directUG} : \mathrm{U} \rightarrow 2^{\mathrm{G}}$, mapping each user to a set of groups, equivalently UGA $\subseteq \mathrm{U} \times \mathrm{G}$
– $\mathrm{directGR} : \mathrm{G} \rightarrow 2^{\mathrm{R}}$, mapping each group to a set of roles, equivalently GRA $\subseteq \mathrm{G} \times \mathrm{R}$
– object-tag $\subseteq$ OB$\times$Tag, relation between object and object tags
– $\mathrm{OBJECT\text{-}PRMS} = 2^{\mathrm{ES} \times (\mathrm{OB} \cup \mathrm{Tag}) \times \mathrm{OP}}$, set of data and service object permissions
– $\mathrm{HS\text{-}PRMS} = 2^{\mathrm{HS} \times \mathrm{OP_{HS}}}$, set of Hadoop services permissions

Permission Assignments
– $\mathrm{PA_{HS}} \subseteq (\mathrm{U} \cup \mathrm{G}) \times$HS-PRMS, mapping entities to Hadoop service permissions. Alternatively, $\mathrm{hs_{prms}}$: (x) $\rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{hs_{prms}}$(x) = {p | (x,p) $\in \mathrm{PA_{HS}}$, x $\in$ (U $\cup$ G)}
– $\mathrm{PA_{ES}} \subseteq (\mathrm{U} \cup \mathrm{G} \cup \mathrm{R}) \times$OBJECT-PRMS, mapping entities to object permissions. Alternatively, $\mathrm{es_{prms}}$: (x) $\rightarrow 2^{\mathrm{OBJECT\text{-}PRMS}}$, defined as $\mathrm{es_{prms}}$(x) = {p | (x,p) $\in \mathrm{PA_{ES}}$, x $\in$ (U $\cup$ G $\cup$ R)}

Hadoop Cross Services Access
– $\mathrm{PA_{HS\text{-}HS}} \subseteq \mathrm{HS} \times$ HS-PRMS, mapping Hadoop service to Hadoop service access. Alternatively, $\mathrm{hs\text{-}hs_{prms}}$: (hs:HS) $\rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{hs\text{-}hs_{prms}}$(hs) = {p | (hs,p) $\in \mathrm{PA_{HS\text{-}HS}}$ }

Effective Roles of Users (Derived Functions)
- $\mathrm{effectiveR} : \mathrm{U} \rightarrow 2^{\mathrm{R}}$, defined as $\mathrm{effectiveR(u)} = \bigcup_{\forall \mathrm{g} \in \mathrm{directUG(u)}} \mathrm{directGR(g)}$

Effective Permissions of User
- $\mathrm{effectiveHS_{prms}} : \mathrm{U} \rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{effectiveHS_{prms}(u)} = \mathrm{hs_{prms}(u)} \cup \bigcup_{\forall \mathrm{g} \in \{\mathrm{directUG(u)}\}} \mathrm{hs_{prms}(g)}$
- $\mathrm{effectiveES_{prms}} : \mathrm{U} \rightarrow 2^{\mathrm{OBJECT\text{-}PRMS}}$, defined as $\mathrm{effectiveES_{prms}(u)} = \mathrm{es_{prms}(u)} \cup \bigcup_{\forall \mathrm{x} \in \{\mathrm{directUG(u)} \cup \mathrm{effectiveR(u)}\}} \mathrm{es_{prms}(x)}$

User Subject
- $\mathrm{userSub} : \mathrm{S} \rightarrow \mathrm{U}$, mapping each subject to its creator user, where the subject gets all the permissions of the creator user.

Ecosystem Service Object Operation Decision
A subject s $\in$ S is allowed to perform an operation op $\in$ OP on an object ob $\in$ OB in ecosystem service es $\in$ ES if the effective object permissions of $\mathrm{userSub(s)}$ include permissions for object ob or for tag t $\in$ Tag associated with object ob. Formally,
(es,ob,op) $\in \mathrm{effectiveES_{prms}}$ ($\mathrm{userSub(s)}$) $\vee$
($\exists$ t) [(ob,t) $\in$ object-tag $\wedge$ (es,t,op) $\in \mathrm{effectiveES_{prms}}$ ($\mathrm{userSub(s)}$)]

application submission or other non-data or object operations. Depending on the type of operations to be performed, a user may require either one or both type of permissions.

A many-to-many relation $\mathrm{PA_{HS}}$ specifies the assignment of $\mathrm{HS\text{-}PRMS}$ to users or groups. In this way a user can be assigned $\mathrm{HS\text{-}PRMS}$ directly or through group membership. $\mathrm{OBJECT\text{-}PRMS}$ can be assigned to users, groups or roles (shown by $\mathrm{PA_{ES}}$). A group can get the object permissions directly or through roles, which will then enable it to the member users. It should be noted that a user may need multiple data object permissions across several data services to operate on a data object. For example, in case of Apache Hive table, besides having permission on the table, a user may be required to have permissions on the underlying data file in HDFS. $\mathrm{PA_{HS\text{-}HS}}$ encapsulates the access requirement between several Hadoop services inside the cluster for task updates or resource monitoring (e.g. communication between DataNodes and NameNode). The effective roles of user are covered by $\mathrm{effectiveR}$ which is union of roles assigned to all member groups. The effective permissions on Hadoop services attained by user (reflected by $\mathrm{effectiveHS_{prms}}$) is the direct permissions on HS and permissions inherited through group membership. The final set of ES object permissions for a user is union of direct permission and permissions assigned through group membership and effective roles as shown in $\mathrm{effectiveES_{prms}}$.

A subject is created by a user as expressed by userSub. It inherits all the permissions assumed by the user to perform actions. In last section of Table 5.1, a subject is allowed to perform operations on objects in ES service depending on either direct permission on objects or permission on tags associated with objects.

5.4 Object-Tagged RBAC Model

In this section we present Object-Tagged Role-Based Access Control model for the Hadoop Ecosystem, which we denote as OT-RBAC. With respect to HeAC model, this model assigns both objects and Hadoop service permissions to users only through roles, consistent with the basic principle of RBAC. The model presents a novel approach for combining attributes and RBAC [144] besides NIST proposed approaches (i.e., Dynamic Roles, Attribute-Centric and Role Centric) [114].

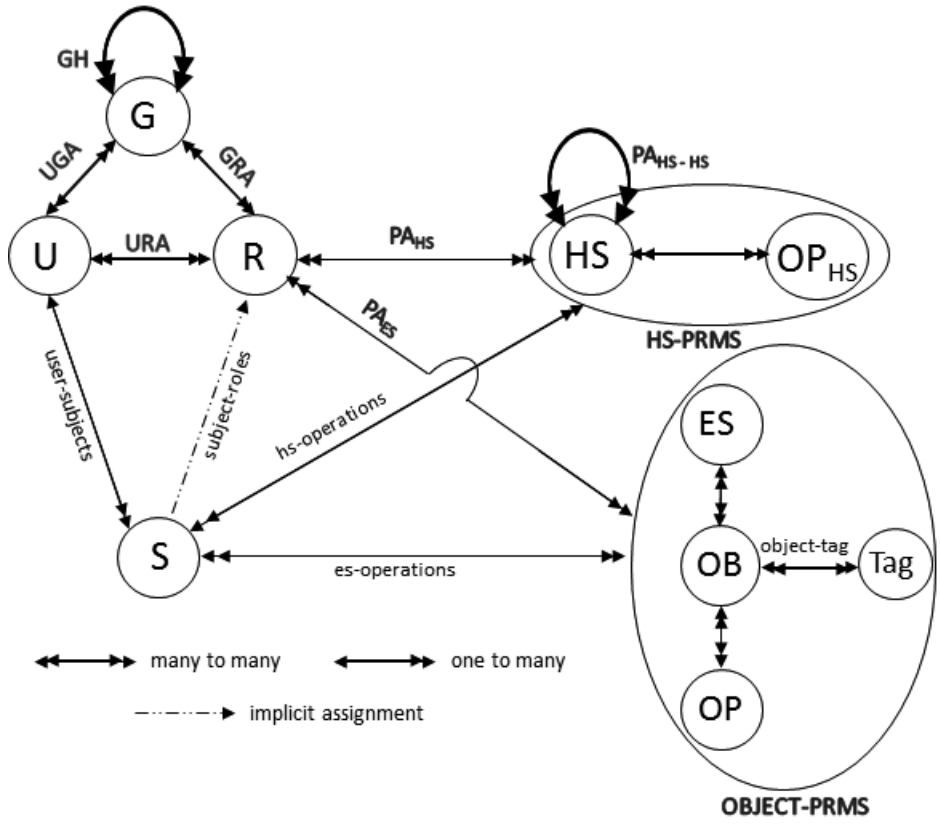


Figure 5.3: Conceptual OT-RBAC Model for Hadoop Ecosystem

Hence the convenient administrative benefits of RBAC, along with a finer-grained attributes authorization using object tags, are incorporated in this model.

5.4.1 Model Definitions

The conceptual model for OT-RBAC is shown in Figure 5.3 followed by formal definitions in Table 5.2 and Table 5.3 (continued). The remainder of this section discusses the new and modified components introduced in OT-RBAC model (marked ** and †† respectively in Table 5.2 and 5.3) with respect to HeAC model. In OT-RBAC model users are directly assigned to multiple roles specified by function $\mathrm{directUR}$. Group hierarchy (GH) is introduced into the system, defined by a partial order relation on G and written as $\succeq_g$. The inheritance of roles is from low to high, i.e., $g_1 \succeq_g g_2$ means g_1 inherits roles from g_2. In such cases, we say g_1 is the senior group and g_2 is the junior group. The $\mathrm{HS\text{-}PRMS}$ and $\mathrm{OBJECT\text{-}PRMS}$ permissions are assigned to roles only, specified by many-to-many relations $\mathrm{PA_{HS}}$ and $\mathrm{PA_{ES}}$ respectively. This is modified with respect

to the original HeAC, where HS-PRMS were assigned to users or groups and OBJECT-PRMS to user, groups or roles also. This reflects the advantage of RBAC model where permissions are allotted or removed from users by granting or revoking their roles. Both OBJECT-PRMS and HS-PRMS can be assigned to same role in the Hadoop ecosystem.

With group hierarchy (GH), the effective roles of a group (expressed by effectiveGR) is the union of direct roles assigned to group and effective roles of all its junior groups. It should be noted that this definition is recursive where the junior-most groups have same direct and effective roles. The effective roles of the user (defined by effectiveR) is then the union of direct user roles and effective roles of the groups to which the user is directly assigned. For example, assuming group Grader is assigned roles Student and Graduate and a senior group TA is assigned to role Doctoral. Then the effective roles of group TA would be Student, Graduate and Doctoral. A user u_1 can be directly assigned to role Staff. If u_1 also becomes a member of group TA, u_1 has the effective roles of Student, Graduate, Doctoral and Staff. The important advantage of user group membership is convenient assignment and removal of multiple roles from users with single administrative operation.

A subject S (similar to sessions in RBAC [72]) created by the user can have some or all of the effective roles of the creator user. The effective permissions available to a subject (expressed by $\text{effectiveES}_{\text{prms}}$ and $\text{effectiveHS}_{\text{prms}}$) will then be the object and Hadoop service permissions assigned to all the effective roles activated by the subject. A subject might need to have multiple permissions to access different services or objects inside Hadoop ecosystem which may result in requiring multiple roles. The prime advantage of OT-RBAC model over HeAC model is the assignment of permissions only to roles instead of assigning directly to users and groups. Further it introduces the concept of group hierarchy which results in roles inheritance and eases administrative responsibilities of the security administrator. Also including group hierarchy makes OT-RBAC model easier to fit into attributes based models where role is one of the other attributes. In such case group hierarchy can be very useful in attributes inheritance offering convenient administration by assigning or removing multiple attributes to users with single administrative operation [153].

Table 5.2: Formal OT-RBAC Model Definitions

Basic Sets and Functions

– U, G, R, S (finite set of users, groups, roles and subjects respectively)
– HS, $\mathrm{OP_{HS}}$ (finite set of Hadoop services and operations respectively)
– ES, OB (finite set of ecosystem services and objects respectively)
– OP, Tag (finite set of object operations and object tags respectively)
– $\mathrm{directUG} : \mathrm{U} \rightarrow 2^{\mathrm{G}}$, mapping each user to a set of groups, equivalently UGA $\subseteq \mathrm{U} \times \mathrm{G}$
– $^{**}\mathrm{directUR} : \mathrm{U} \rightarrow 2^{\mathrm{R}}$, mapping each user to a set of roles, equivalently URA $\subseteq \mathrm{U} \times \mathrm{R}$
– $\mathrm{directGR} : \mathrm{G} \rightarrow 2^{\mathrm{R}}$, mapping each group to a set of roles, equivalently GRA $\subseteq \mathrm{G} \times \mathrm{R}$
– $^{**}\mathrm{GH} \subseteq \mathrm{G} \times \mathrm{G}$, a partial order relation $\succeq_g$ on G
– $\mathrm{object\text{-}tag} \subseteq \mathrm{OB} \times \mathrm{Tag}$, relation between object and object tags
– $\mathrm{OBJECT\text{-}PRMS} = 2^{\mathrm{ES} \times (\mathrm{OB} \cup \mathrm{Tag}) \times \mathrm{OP}}$, set of data and service object permissions
– $\mathrm{HS\text{-}PRMS} = 2^{\mathrm{HS} \times \mathrm{OP_{HS}}}$, set of Hadoop services permissions

††Role Permission Assignments

– $\mathrm{PA_{HS}} \subseteq \mathrm{R} \times \mathrm{HS\text{-}PRMS}$, mapping roles to Hadoop service permissions. Alternatively, $\mathrm{hs_{prms}} : (\mathrm{r}{:}\mathrm{R}) \rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{hs_{prms}}(\mathrm{r}) = \{\mathrm{p} \mid (\mathrm{r},\mathrm{p}) \in \mathrm{PA_{HS}}\}$
– $\mathrm{PA_{ES}} \subseteq \mathrm{R} \times \mathrm{OBJECT\text{-}PRMS}$, mapping roles to object permissions. Alternatively, $\mathrm{es_{prms}} : (\mathrm{r}{:}\mathrm{R}) \rightarrow 2^{\mathrm{OBJECT\text{-}PRMS}}$, defined as $\mathrm{es_{prms}}(\mathrm{r}) = \{\mathrm{p} \mid (\mathrm{r},\mathrm{p}) \in \mathrm{PA_{ES}}\}$

Hadoop Cross Services Access

– $\mathrm{PA_{HS\text{-}HS}} \subseteq \mathrm{HS} \times \mathrm{HS\text{-}PRMS}$, mapping Hadoop service to Hadoop service access. Alternatively, $\mathrm{hs\text{-}hs_{prms}} : (\mathrm{hs}{:}\mathrm{HS}) \rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{hs\text{-}hs_{prms}}(\mathrm{hs}) = \{\mathrm{p} \mid (\mathrm{hs},\mathrm{p}) \in \mathrm{PA_{HS\text{-}HS}}\}$

††Effective Roles of Users (Derived Functions)

- $\mathrm{effectiveGR} : \mathrm{G} \rightarrow 2^{\mathrm{R}}$, defined as $\mathrm{effectiveGR}(\mathrm{g_i}) = \mathrm{directGR}(\mathrm{g_i}) \cup \left(\bigcup_{\forall \mathrm{g} \in \{\mathrm{g_j} \mid \mathrm{g_i} \succeq_g \mathrm{g_j}\}} \mathrm{effectiveGR}(\mathrm{g}) \right)$
- $\mathrm{effectiveR} : \mathrm{U} \rightarrow 2^{\mathrm{R}}$, defined as $\mathrm{effectiveR}(\mathrm{u}) = \mathrm{directUR}(\mathrm{u}) \cup \left(\bigcup_{\forall \mathrm{g} \in \mathrm{directUG}(\mathrm{u})} \mathrm{effectiveGR}(\mathrm{g}) \right)$

††Effective Roles and Permissions of Subjects

- $\mathrm{userSub} : \mathrm{S} \rightarrow \mathrm{U}$, mapping each subject to its creator user
- $\mathrm{effectiveR} : \mathrm{S} \rightarrow 2^{\mathrm{R}}$, mapping of subject s to a set of roles. It is required that : $\mathrm{effectiveR}(\mathrm{s}) \subseteq \mathrm{effectiveR}(\mathrm{userSub}(\mathrm{s}))$
- $\mathrm{effectiveHS_{prms}} : \mathrm{S} \rightarrow 2^{\mathrm{HS\text{-}PRMS}}$, defined as $\mathrm{effectiveHS_{prms}}(\mathrm{s}) = \bigcup_{\forall \mathrm{r} \in \mathrm{effectiveR}(\mathrm{s})} \mathrm{hs_{prms}}(\mathrm{r})$
- $\mathrm{effectiveES_{prms}} : \mathrm{S} \rightarrow 2^{\mathrm{OBJECT\text{-}PRMS}}$, defined as $\mathrm{effectiveES_{prms}}(\mathrm{s}) = \bigcup_{\forall \mathrm{r} \in \mathrm{effectiveR}(\mathrm{s})} \mathrm{es_{prms}}(\mathrm{r})$

** and †† highlight new and modified components respectively with respect to HeAC

Table 5.3: Formal OT-RBAC Model Definitions (Continued)

Ecosystem Service Object Operation Decision A subject $s \in S$ is allowed to perform an operation op $\in$ OP on an object $ob \in OB$ in ecosystem service es $\in$ ES if the effective object permissions of subject s include permissions to object ob or to tag t $\in$ Tag associated with object ob. Formally, (es,ob,op) $\in \text{effectiveES}_{\text{prms}}(s) \vee$ ($\exists$ t) [(ob,t) $\in$ object-tag $\wedge$ (es,t,op) $\in \text{effectiveES}_{\text{prms}}(s)$]

** and †† highlight new and modified components respectively with respect to HeAC

The proposed OT-RBAC model presents a novel approach for adding attributes to RBAC (besides NIST strategies [114]), by introducing object tags. The model represents object permissions (OBJECT-PRMS) as union of permissions on attribute values (reflected as tags) associated with objects and regular permissions as discussed in RBAC [144]. In the following section, we propose an implementation approach for OT-RBAC using open-source Apache Ranger.

5.4.2 Implementation and Evaluation

One approach to implement OT-RBAC model is by extending open-source Apache Ranger which provides centralized security administration to multiple Hadoop ecosystem services. It offers REST API to create security policies which are enforced using plugins appended to each secured service. These plugins intercept a user access request, and check against policies cached sporadically from policy server to make access decisions. Apache Ranger 0.5 and above provide extensible framework to add new authorization functionalities by offering context enricher and condition evaluator hooks. Context enricher is a Java class which appends user access request with additional information used for policy evaluation. Condition evaluator enables a security architect to add custom conditions to policies. These hooks can be used to extend plugins to enforce OT-RBAC in Hadoop ecosystem.

Proposed Apache Ranger architecture for Hive service authorization is shown in Figure 5.4. Users and groups are stored in Lightweight Directory Access Protocol (LDAP), which are synced to Ranger policy manager to create policies. A text file is added which stores current users to roles

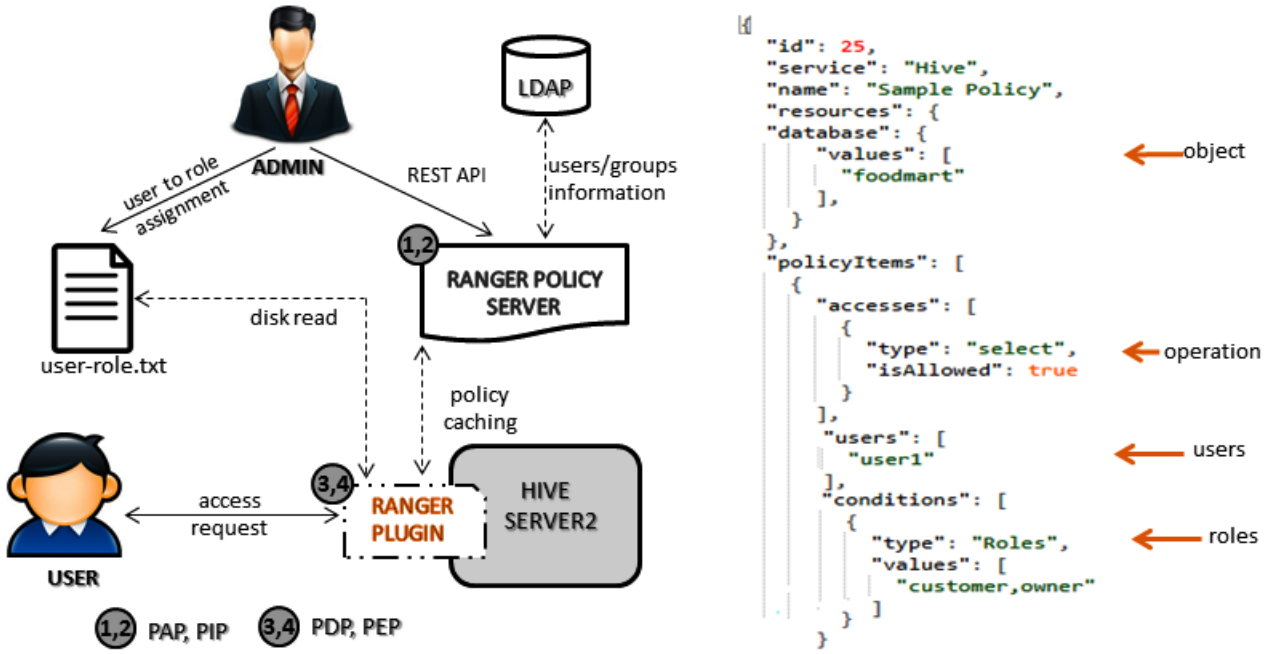


Figure 5.4: Proposed Implementation in Apache Ranger and Sample JSON Policy

assignment. This file is used by context enricher implemented, to add roles of user to access request along with objects and actions. A condition evaluator should also be implemented to include roles in policy used for evaluation. A sample policy in JSON format is shown in Figure 5.4. This policy includes roles in condition which specifies the roles allowed to perform select operation on table foodmart. Hive service definition should be updated with new context and condition hooks information using REST API. Access decision and enforcement is done in Ranger plugin embedded with Hive service whereas policy administration and information is through central policy server as shown in Figure 5.4. Similar implementation approach can be adopted in other ecosystem services also. This proposed implementation requires roles addition at two places, one in text file and other in policy conditions which requires extra effort by administrator.

We simulated the effect of roles using groups in Apache Ranger where a set of users were assigned permissions directly and another set via group membership. Users and groups were synced from Active Directory (FreeIPA v3.0.0) and access policy for Hive service was created using Ranger API. A 6 node Hadoop cluster (with CentOS6 and 1Gbps network speed) each having 500GB disk space, 32GB RAM and 8 CPU Intel 2.5GHz was set up, with a separate client node

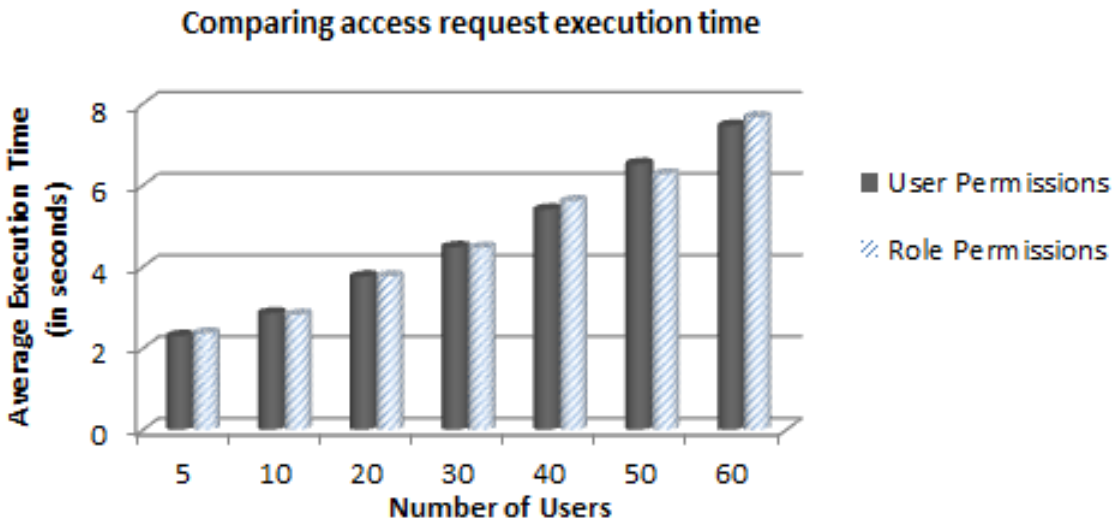


Figure 5.5: Performance Evaluation

having 32 CPU Intel Core i7 2GHz to send access requests. As shown in Figure 5.5, the average execution time for user access request is same (with marginal reading error) for users getting direct permissions compared to users which are assigned permissions through roles. Hence, in can be conjectured that OT-RBAC model provides convenient permission-assignment benefits of RBAC without impacting execution time of user requests.

5.5 Attributes Based Extensions to OT-RBAC

This section outlines some approaches for adding attributes in OT-RBAC model to achieve finer-grained access control. OT-RBAC model incorporates tags for objects, which is further generalized by introducing set of object attributes along with attributes for other entities. As shown in Figure 5.6, UA is a set of attributes for users and groups, and OA is a set of attributes for data and service objects. HSA and ESA are set of attributes for HS and ES. An attribute is a function which takes as input an entity and returns values from a specified range [105]. Attribute-based authorization policies are used to determine access permissions of users on services and objects. With group hierarchy, senior groups inherit attributes from junior groups [88], and a user assigned to senior groups gets all attributes of group besides its direct attributes. A set of environment attributes is

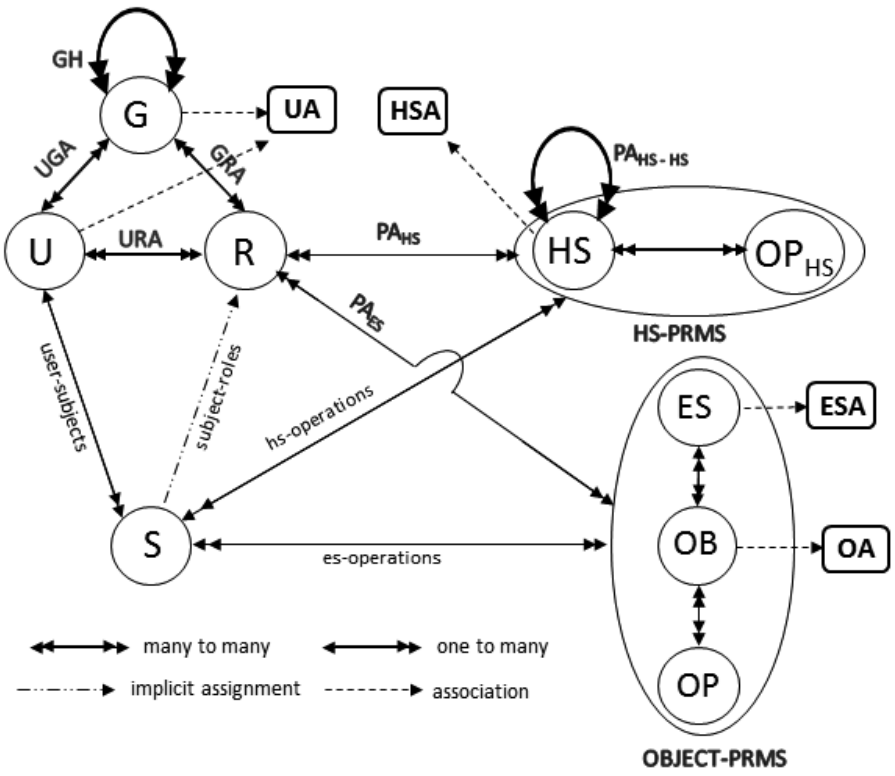


Figure 5.6: Adding Attributes to OT-RBAC Model

also added to incorporate contextual information (like access time, threat level) in policies.

We outline how an attribute enhanced OT-RBAC model, along the lines of Figure 5.6, can incorporate NIST proposed strategies [114] for adding attributes in RBAC, i.e., Dynamic Roles, Attribute Centric and Role Centric. We discuss these in context of objects permissions assignment. These approaches can be similarly applied to Hadoop services permissions assignment also.

5.5.1 Dynamic Roles

Dynamic Roles approach considers user and environment attributes to determine roles of a user. This automated approach require rules defined using a policy language [36] composed of attributes and resulting roles. The roles of the user will change based on the user's current attributes as well as current environment attributes. As shown in Figure 5.7, OT-RBAC model can be configured to achieve dynamic roles assignment to users based on the direct or inherited attributes through group memberships [88]. We can further extend the use of attributes for dynamic permissions assignment

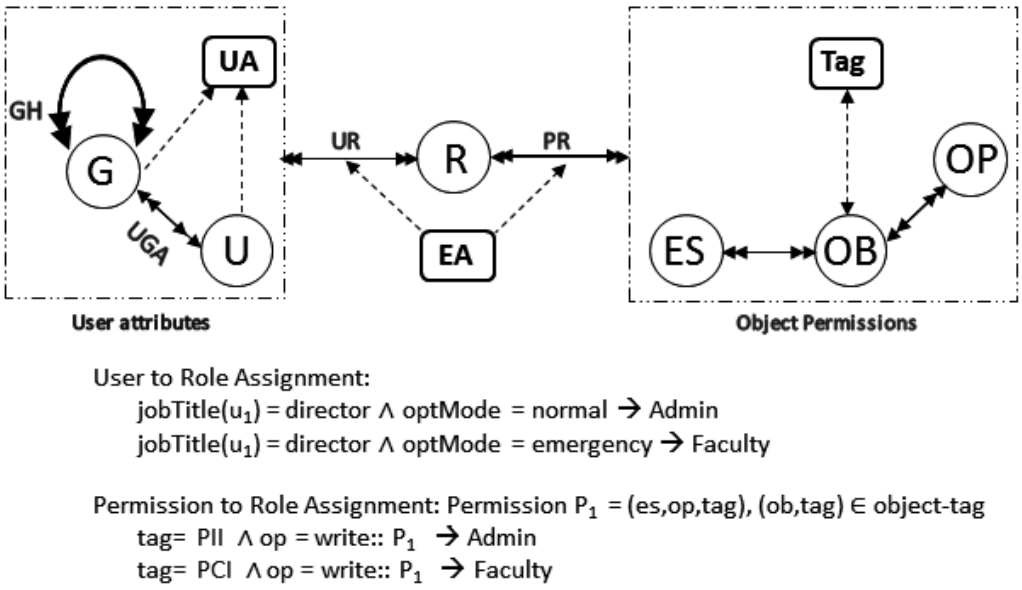


Figure 5.7: Dynamic Roles and Object Permissions in OT-RBAC

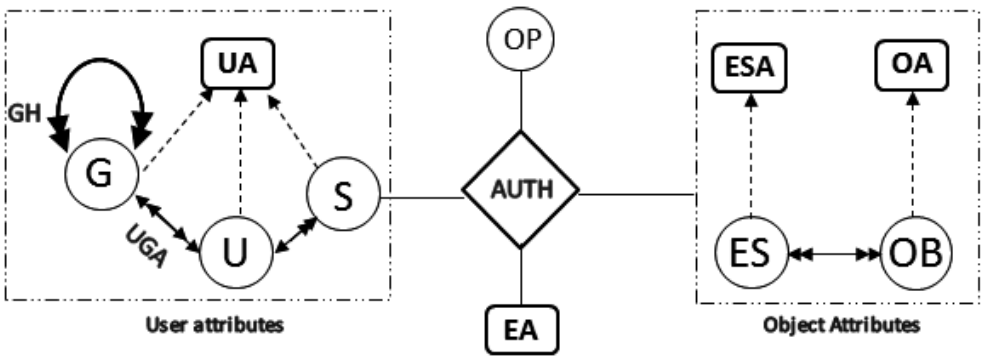
to roles based on object tags, environment attribute values and operations.

As shown in Figure 5.7, a user u_1 with attribute jobTitle value director and environment attribute optMode value normal can be assigned an Admin role, which can change to role Faculty when attribute optMode changes to emergency value. Similarly, a permission containing operation write on object ob with tag value PII can be assigned to role Admin which can further change to role Faculty when tag changes to PCI.

5.5.2 Attribute Centric

In this approach, access decision is based on attributes of entities (role is also an attribute) where authorization policies comprise attributes of subjects, objects or environment [99, 105, 178]. To configure OT-RBAC with attribute centric strategy, boolean authorization functions are defined using propositional logic formula for each operation in OP which specify policy if subject s can perform operation op on object ob in ecosystem service es under some environment attributes.

As shown in Figure 5.8, authorization policy is defined stating that subject s with effective attribute jobTitle value director is allowed to perform write on object ob with attribute tag value PII in ecosystem service es with name hdfs when environment attribute optMode is normal. It



$Authorization_{write}$ (s:S, es:ES, ob:OB) :: $effective_{jobTitle}$(s) = director ∧ access(s,es) = True ∧ name(es) = hdfs ∧ tag(ob)= PII ∧ optMode = normal

$Authorization_{read}$ (s:S, es:ES, ob:OB) :: $effective_{jobTitle}$(s) = professor ∧ access(s,es) = True ∧ name(es) = hdfs ∧ tag(ob)= PCI ∧ optMode = emergency.

Figure 5.8: Attribute Centric approach in OT-RBAC

should be noted that object ob must belong to ecosystem service es and subject must be allowed to access es (expressed by access(s,es)) before performing any operation on object in es. Similar authorization policy for read operation can be defined by administrators.

5.5.3 Role Centric

In this approach the maximum permissions (avail_prms) are assigned to user through roles assignment (similar to RBAC [144]) but the final set of permissions (final_prms) is dependent on the attributes of entities. Permission Filtering boolean functions are defined based on the attributes, which are checked for each permission in avail_prms set available to users via roles, to determine the final_prms set assigned to the users as discussed in [108].

Assume user u_1 assigned to role Admin then u_1 gets permissions (avail_prms) of writing to hdfs service file customer and reading a file having PII tag. These permissions are checked against filter functions selected using target functions discussed in [108]. As shown in Figure 5.9, filter function FAdmin1 is invoked to check if first permission is in final_prms set. The function checks if creator user of s has jobTitle attribute with value director and optMode is normal to avail this permission. If it returns true, the permission will be included in final set (final_prms). Similar filter function can be called for other permissions also.

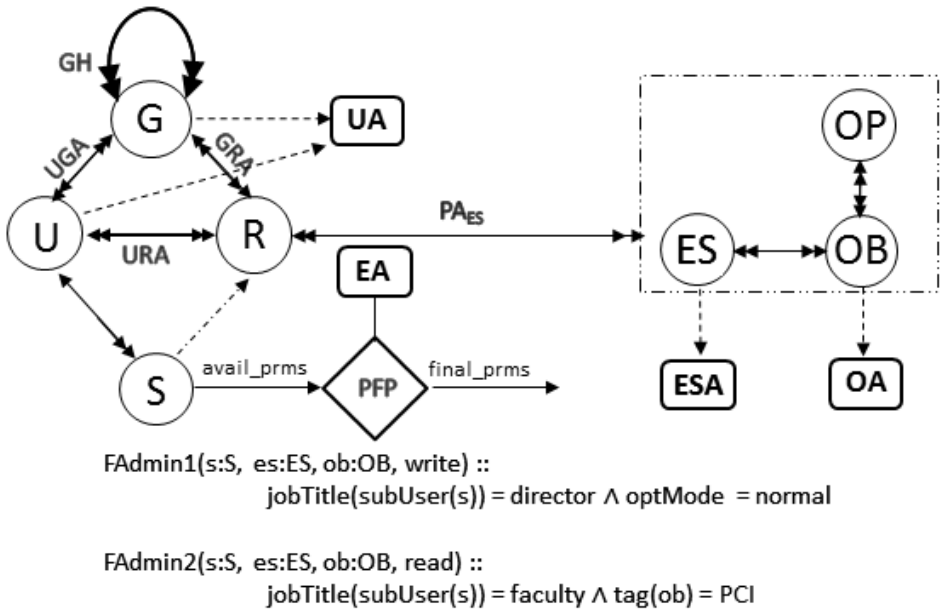


Figure 5.9: Role Centric approach in OT-RBAC

5.6 Attribute Based Access Control for Hadoop Ecosystem

Attribute based access control is known to offer flexible fine grained authorization capabilities by introducing the characteristics of subjects, objects, environment or context in access control decision. Such mechanisms are required in complex distributed systems like Hadoop where multi-tenant data lake is being accessed at varied data granularity levels by multiple users at different time, locations and conditions. Real-world use-cases like a user allowed to access data only from a particular location or IP address or at a specific time are very common and most conveniently addressed by attribute based systems. We will now define the attribute based access control model for Hadoop ecosystem, referred as HeABAC and shown in Figure 5.10. The complete formalization of HeABAC model is given in Table 5.4.

5.6.1 HeABAC Model Definitions

The basic sets of HeABAC model involve the previously defined access control components — Users (U), Groups (G), Subjects (S), Hadoop Services (HS), Ecosystem Services (ES), Data and service objects (OB), Operations on objects in Ecosystem Services (OP), Operations on Hadoop

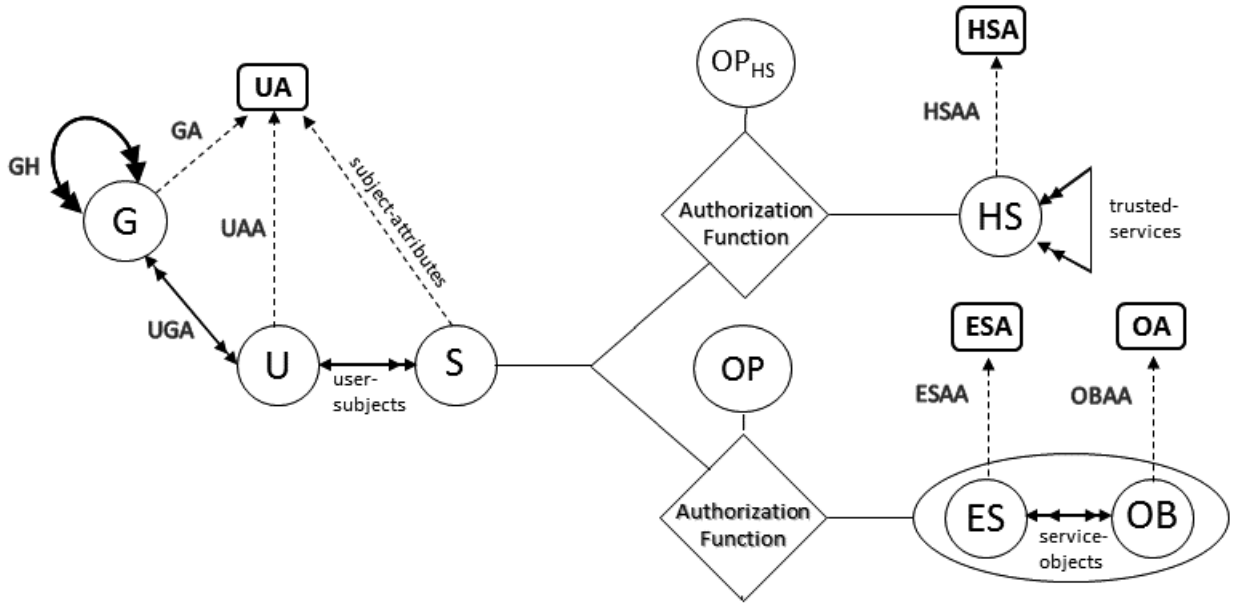


Figure 5.10: The Conceptual HeABAC Model for Hadoop Ecosystem

Services ($\mathrm{OP_{HS}}$), as elaborated in Section 5.3 and stated in Table 5.4. Some of these entities have characteristics which are used in access control decision and are expressed as their attributes. User attributes (UA) is the set of user attributes for users, groups and subjects. Object attributes (OA) is the set of object attributes assigned to data and service objects (OB). Ecosystem service attribute (ESA) and Hadoop service attribute (HSA) are the set of attribute functions which can be assigned to Ecosystem services (ES) and Hadoop Services (HS) respectively. Users, groups, Hadoop or ecosystem services and objects can be assigned attribute values directly for an attribute function att (in their respective sets) from the set of atomic values in the range, denoted by Range(att). Attribute functions in UA are required to be only set valued whereas for other sets OA, ESA and HSA both set and atomic valued functions are allowed. Each attribute function in UA, denoted by $\mathrm{att_u}$, will map a user or group to a set of values in power set of Range($\mathrm{att_u}$). Similarly, attribute functions in OA, ESA and HSA map OB, ES and HS respectively to one or subset of attribute values from the range depending on atomic or set valued attribute type as shown in Table 5.4.

Users are assigned to multiple groups (defined by many to many function directUG) to achieve simplified administration of attributes. When a user is made member to a group, the user inherits all attributes of the group, whereby multiple attributes can be assigned or removed from a user by

Table 5.4: Formal ABAC Model Definitions for Hadoop Ecosystem

Basic Sets and Functions

– U, G, S are finite sets of users, groups and subjects respectively.
– HS, ES are finite sets of Hadoop services and ecosystem services respectively.
– OB, OP are finite sets of objects and object operations respectively.
– $\mathrm{OP_{HS}}$ is a finite set of operations on Hadoop services.
– UA, OA are finite sets of user and object attribute functions respectively.
– ESA, HSA are finite sets of ecosystem and Hadoop service attribute functions respectively.
– For each attribute att in UA ∪ OA ∪ ESA ∪ HSA, Range(att) is a finite set of atomic values.
– attType: UA = {set}, defines user attributes to be set valued only.
– attType: OA ∪ ESA ∪ HSA = {set, atomic}, defines other attributes to be set or atomic.
– For each attribute $\mathrm{att_u}$ in UA, $\mathrm{att_u} : \mathrm{U} \cup \mathrm{G} \rightarrow 2^{\mathrm{Range(att_u)}}$ mapping each user or group to a set of attribute values in $\mathrm{Range(att_u)}$.
– Each attribute $\mathrm{att_{ob}}$ in OA maps objects in OB to attribute values.
Formally, $\mathrm{att_{ob}} : \mathrm{OB} \rightarrow \begin{cases} \mathrm{Range(att_{ob})} & \text{if attType(att}_{\mathrm{ob}}\text{) = atomic} \\ 2^{\mathrm{Range(att_{ob})}} & \text{if attType(att}_{\mathrm{ob}}\text{) = set} \end{cases}$
– Each attribute $\mathrm{att_{es}}$ in ESA maps services in ES to attribute values.
Formally, $\mathrm{att_{es}} : \mathrm{ES} \rightarrow \begin{cases} \mathrm{Range(att_{es})} & \text{if attType(att}_{\mathrm{es}}\text{) = atomic} \\ 2^{\mathrm{Range(att_{es})}} & \text{if attType(att}_{\mathrm{es}}\text{) = set} \end{cases}$
– Each attribute $\mathrm{att_{hs}}$ in HSA maps services in HS to attribute values.
Formally, $\mathrm{att_{hs}} : \mathrm{HS} \rightarrow \begin{cases} \mathrm{Range(att_{hs})} & \text{if attType(att}_{\mathrm{hs}}\text{) = atomic} \\ 2^{\mathrm{Range(att_{hs})}} & \text{if attType(att}_{\mathrm{hs}}\text{) = set} \end{cases}$
– $\mathrm{directUG} : \mathrm{U} \rightarrow 2^{\mathrm{G}}$, mapping each user to a set of groups, equivalently UGA $\subseteq \mathrm{U} \times \mathrm{G}$
– GH $\subseteq$ G×G, a partial order relation $\succeq_g$ on G

Effective Attributes of Users (Derived Functions)

– For each attribute $\mathrm{att_u}$ in UA,
- $\mathrm{effectiveG_{att_u}} : \mathrm{G} \rightarrow 2^{\mathrm{Range(att_u)}}$, defined as
$\mathrm{effectiveG_{att_u}(g_i)} = \mathrm{att_u(g_i)} \cup (\bigcup_{\mathrm{g} \in \{\mathrm{g_j} | \mathrm{g_i} \succeq_g \mathrm{g_j}\}} \mathrm{effectiveG_{att_u}(g)})$.

– For each attribute $\mathrm{att_u}$ in UA,
- $\mathrm{effectiveU_{att_u}} : \mathrm{U} \rightarrow 2^{\mathrm{Range(att_u)}}$, defined as
$\mathrm{effectiveU_{att_u}(u)} = \mathrm{att_u(u)} \cup (\bigcup_{\mathrm{g} \in \mathrm{directUG(u)}} \mathrm{effectiveG_{att_u}(g)})$.

Effective Attributes of Subjects

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

Cross Hadoop Services Trust

– trusted-services : $\mathrm{HS} \rightarrow 2^{\mathrm{HS}}$ is a required function to map Hadoop services to a set of trusted Hadoop services. Equivalently, relation service trust written as $\trianglelefteq \subseteq \mathrm{HS} \times \mathrm{HS}$, where $\mathrm{hs_a} \trianglelefteq \mathrm{hs_b}$ iff $\mathrm{hs_b} \in$ trusted-services($\mathrm{hs_a}$), meaning trustee service $\mathrm{hs_b}$ is trusted by trustor service $\mathrm{hs_a}$ and Hadoop service $\mathrm{hs_b}$ can access service $\mathrm{hs_a}$.

Table 5.5: Formal ABAC Model Definitions for Hadoop Ecosystem (Continued)

Authorization Functions

– Service Authorization Function: For each op $\in \mathrm{OP_{HS}}$, $\mathrm{Authorization_{op}(s : S, sr : HS \cup ES)}$ is a propositional logic formula returning true or false, which is defined using the following policy language:

- $\alpha ::= \alpha \wedge \alpha \mid \alpha \vee \alpha \mid (\alpha) \mid \neg\alpha \mid \exists\ \mathrm{x} \in \mathrm{set}.\alpha \mid \forall\ \mathrm{x} \in \mathrm{set}.\alpha \mid \mathrm{set} \triangle \mathrm{set} \mid\ \mathrm{atomic} \in \mathrm{set} \mid \mathrm{atomic} \notin \mathrm{set}$
- $\triangle ::= \subset \mid \subseteq \mid \not\subseteq \mid \cap \mid \cup$
- $\mathrm{set} ::= \mathrm{effective_{att_u}(s)} \mid \mathrm{att_{sr}(sr)}$ for $\mathrm{att_u} \in$ UA, sr $\in$ ES $\cup$ HS, attType(att) = set
- $\mathrm{atomic} ::= \mathrm{att_{sr}(sr)} \mid \mathrm{value}$ sr $\in$ ES $\cup$ HS, attType(att) = atomic

– Data or Service Objects Authorization Function: For each op $\in$ OP on objects ob $\in$ OB belonging to ecosystem service es $\in$ ES, $\mathrm{Authorization_{op}(s : S, es : ES, ob : OB)}$ is a propositional logic formula returning true or false, which is defined using the policy language stated above with following changes

- $\mathrm{set} ::= \mathrm{effective_{att_u}(s)} \mid \mathrm{att_{es}(es)} \mid \mathrm{att_{ob}(ob)}$ for $\mathrm{att_u} \in$ UA, attType(att) = set
- $\mathrm{atomic} ::= \mathrm{att_{es}(es)} \mid \mathrm{att_{ob}(ob)} \mid \mathrm{value}$ attType(att) = atomic

Access Operation Decision

– A subject $\mathrm{s} \in$ S is allowed to perform an operation op $\in \mathrm{OP_{HS}}$ on a service sr $\in$ ES $\cup$ HS if the effective attributes of subject s and attributes of services satisfy policies stated in $\mathrm{Authorization_{op}(s : S, sr : HS \cup ES)}$. Formally, $\mathrm{Authorization_{op}(s : S, sr : HS \cup ES)}$ = True.

– A subject $\mathrm{s} \in$ S is allowed to perform an operation op $\in$ OP on an object $\mathrm{ob} \in$ OB in ecosystem service es $\in$ ES if subject is allowed to access services es and the effective attributes of subject s, the attributes of object ob and service es satisfy policies in $\mathrm{Authorization_{op}(s : S, es : ES, ob : OB)}$. Formally, $\mathrm{Authorization_{access}(s : S, es : ES)}$ = True $\wedge$ $\mathrm{Authorization_{op}(s : S, es : ES, ob : OB)}$ = True.

just a single administrative action. Further, group hierarchy (GH) also exists in the system (shown as self loop on G), defined using a partial order relation on G and denoted by $\succeq_g$, where $g_1 \succeq_g g_2$ signifies g_1 is senior to g_2 and g_1 will inherit all the attributes of g_2. Therefore, for attribute $\mathrm{att_u}$, the effective values for group g_1 is the union of values directly assigned to g_1 for $\mathrm{att_u}$ and the effective values of $\mathrm{att_u}$ for all the junior groups to g_1, as defined by $\mathrm{effectiveG_{att_u}}(g_1)$. The effective attribute values of a user for attribute $\mathrm{att_u}$ will then be the directly assigned values to user for $\mathrm{att_u}$ and the effective attribute values of attribute $\mathrm{att_u}$ for all the groups to which user is directly assigned. For example, if group g_1 has attribute role with value Chair, and a junior group g_2 has role with value Faculty, then the effective values of attribute role for group g_1 will be Chair and Faculty. Further, when a user is assigned to group g_1, it will inherit all values of attribute role (i.e. Chair and

Faculty) besides the values directly assigned to user, as further elaborated in [88, 153]. A subject which is created by the user (denoted by function userSub) inherits subset or all the values of effective attributes of the creator user as stated by $\mathrm{effectiveU}_{\mathrm{att_u}}(\mathrm{s}) \subseteq \mathrm{effectiveU}_{\mathrm{att_u}}(\mathrm{userSub(s)})$. These values can change with time but must not exceed values of the creator user.

In Hadoop ecosystem, several Hadoop services interact or access other Hadoop services for task updates or cluster resource status (like HDFS NameNode and DataNode or YARN ResourceManager and ApplicationMaster). We refer to this type of interaction as Cross Hadoop Service Trust in Table 5.4 (stated as trusted-services), determining which Hadoop services are allowed to access other services. In this cross service relation, we introduce the notion of Cross Hadoop service trust as a many to many relation where $\mathrm{HS_A} \trianglelefteq \mathrm{HS_B}$ denotes that $\mathrm{HS_B}$ is a trusted service by $\mathrm{HS_A}$ and therefore, $\mathrm{HS_B}$ is allowed to access or interact with $\mathrm{HS_A}$. In this case $\mathrm{HS_B}$ is a trustee service and $\mathrm{HS_A}$ is a trustor service, and trust relation existence is controlled by $\mathrm{HS_A}$. This trust relation obviates the need to specify ACLs as done in HeAC model. For example, a service level authorization ACL `security.datanode.protocol.acl` controls which DataNodes are allowed to communicate with NameNodes. In such cases, a DataNode running as datanode1 can access service NameNode namenode1, if cross service trust relation is established between them i.e. namenode1 $\trianglelefteq$ datanode1. Different types of cross Hadoop service trust relations can exist in the system which are discussed further in the next subsection.

5.6.2 Concept of Cross Hadoop Services Trust

Cross Hadoop service trust determines which two Hadoop services can interact with each other. Our definition of trust relations are primarily unidirectional and involves only two Hadoop services, a trustor and trustee. We assert that trust is established unilaterally by the trustor, and can only be revoked or modified by the trustor.

The cross Hadoop trust relation $\trianglelefteq$ is a binary relation established between trustor and trustee services. A service can be a trustor in one relation and trustee in another. This relation has the following defining properties, for Hadoop services $\mathrm{hs_a}$, $\mathrm{hs_b}$, $\mathrm{hs_c} \in$ HS:

- **Reflexive:** A Hadoop service must always trust itself, meaning $\mathrm{hs_a} \trianglelefteq \mathrm{hs_a}$.
- **Non Transitive:** The Cross Hadoop service trust relation is always defined by the trustor and cannot be inferred from any indirect combinations of other trust relationships i.e. $\mathrm{hs_a} \trianglelefteq \mathrm{hs_b} \wedge \mathrm{hs_b} \trianglelefteq \mathrm{hs_c} \not\Rightarrow \mathrm{hs_a} \trianglelefteq \mathrm{hs_c}$.
- **Non Symmetric and Non Asymmetric:** This characteristic states that the trust relation is always unidirectional and is also independent in each direction i.e. $\mathrm{hs_a} \trianglelefteq \mathrm{hs_b} \not\Rightarrow \mathrm{hs_b} \trianglelefteq \mathrm{hs_a}$ and $\mathrm{hs_a} \trianglelefteq \mathrm{hs_b} \wedge \mathrm{hs_b} \trianglelefteq \mathrm{hs_a} \not\Rightarrow \mathrm{hs_a} \equiv \mathrm{hs_b}$.

We will now identify and discuss four potential types of trust relations to enable cross Hadoop services access control. The type of relation is determined by who controls the existence of trust relation and who controls access to the service. These types of relations are analogous and adapted from the trust relation types discussed in [137, 160–162].

Type-α. In this relation, trustor grants access to trustee and the relation is controlled (exists or created) by the trustor. For example, if $\mathrm{hs_a} \trianglelefteq_\alpha \mathrm{hs_b}$, then Hadoop service $\mathrm{hs_a}$ authorizes service $\mathrm{hs_b}$ to access $\mathrm{hs_a}$, and the relation is controlled by $\mathrm{hs_a}$ and service access is also controlled by $\mathrm{hs_a}$. This type of relation are most intuitive and for simplicity, is used in our HeABAC model.

Type-β. In this relation, trustee grants access to trustor and the relation is still controlled by trustor. For example, if $\mathrm{hs_a} \trianglelefteq_\beta \mathrm{hs_b}$, then Hadoop service $\mathrm{hs_b}$ authorizes service $\mathrm{hs_a}$ to access $\mathrm{hs_b}$, and the relation is controlled by $\mathrm{hs_a}$ and service access is also controlled by $\mathrm{hs_a}$ without the consent from service $\mathrm{hs_b}$.

Type-γ. In this relation, trustee controls access of trustor and the relation is still controlled by trustor. For example, if $\mathrm{hs_a} \trianglelefteq_\gamma \mathrm{hs_b}$, then Hadoop service $\mathrm{hs_b}$ authorizes service $\mathrm{hs_a}$ to access $\mathrm{hs_b}$. Here the relation is controlled by $\mathrm{hs_a}$ but service access is controlled by Hadoop service $\mathrm{hs_b}$.

Type-δ. In this relation, trustee takes access from trustor by approving or denying access. For example, if $\mathrm{hs_a} \trianglelefteq_\delta \mathrm{hs_b}$, then Hadoop service $\mathrm{hs_a}$ authorizes service $\mathrm{hs_b}$ to access $\mathrm{hs_a}$, and the relation existence is controlled by $\mathrm{hs_a}$ but the service access is controlled by the consent of $\mathrm{hs_b}$.

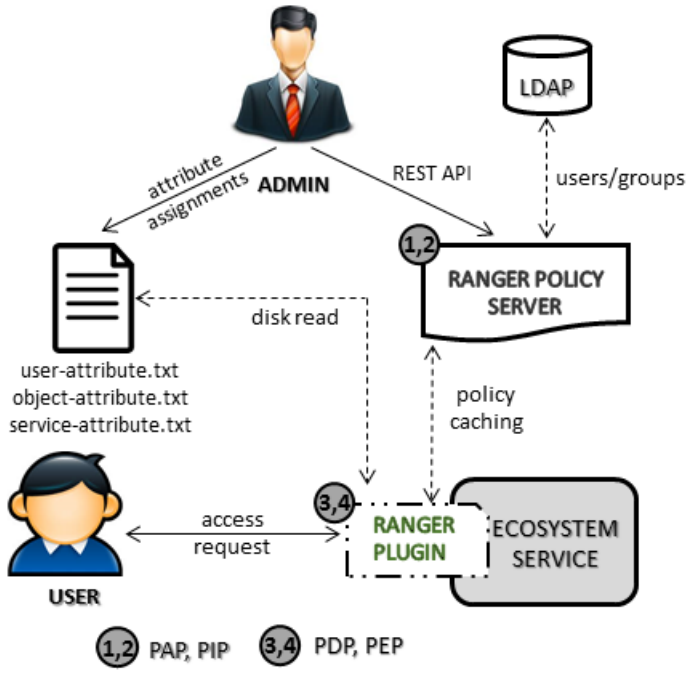


Figure 5.11: Proposed HeABAC Implementation in Apache Ranger

5.6.3 HeABAC Implementation Approach

Apache Ranger and Apache Sentry are two dominant open-source security projects in Hadoop ecosystem which are focussed in offering authorization and access control capabilities in several ecosystem projects including Apache Hive, HBase, Kafka etc. Both Apache Ranger and Sentry provides security plugins which are attached to different ecosystem services which needs to be protected. Every access request by a user is intercepted by these plugins which check the security policies defined by the administrator using REST API or user interface, to decide and enforce access control decisions. Apache Ranger currently offers some fine grained extensions where attributes of a user are embedded into the access request using context enricher. These context enrichers are Java class which enriches the request of the user with extra information which is used in the security policy conditions to approve or deny request. These conditions are evaluated using condition evaluators which are also defined in Apache Ranger. For example, is a user Alice wants to access an object obj1 but the policy specifies that user Alice can only access resource obj1 if time is after 10 pm, then, the context enricher will add on the current time into the access request

of the user Alice and the condition evaluator will check if the access request complies with the time condition specified in the policy.

Our proposed implementation of HeABAC extends the current context enricher and condition evaluators in Apache Ranger. We propose that context enricher will not only be used for enriching user information but also for services and objects in the access request. As shown in Figure 5.11, the security administrator will add text files for different users, objects and services into the system with their relevant attributes. These files will then be used by context enricher implemented, which will add attributes of users, services and objects in the access request. Similarly, condition evaluators also need to be extended to incorporate the attributes of objects and services in policies, which will be also evaluated when the enriched access request with attributes is checked against a defined security policy. Here, the administration of policies is done through the central policy server while the decisions and enforcement are made by Apache Ranger security plugins attached with the individual services as shown in Figure 5.11.

5.7 Use Cases and HeABAC Application

In this section we illustrate some important use-cases to emphasise the use and benefits of fine grained and flexible attribute based access control in Hadoop ecosystem. These use-cases will reflect real world scenarios and will cover different access control requirements as discussed in earlier sections of this chapter. In these use-cases we consider that users have already been authenticated by some external mechanisms and data is already ingested into the Hadoop system before access control comes into enforcement.

Internet of Things is a growing buzz among different businesses, and several enterprises are harnessing the potential it offers. It has spread itself to different spheres of our life including health, smart homes and more recently to smart city and transportation. Vehicular Internet of Things is the future where vehicles and road infrastructure will be communicating with each other. These vehicles will be generating lot of data ranging from car sensor readings, road conditions or even driver health vitals to be analysed by different stake-holders for better and life saving services to

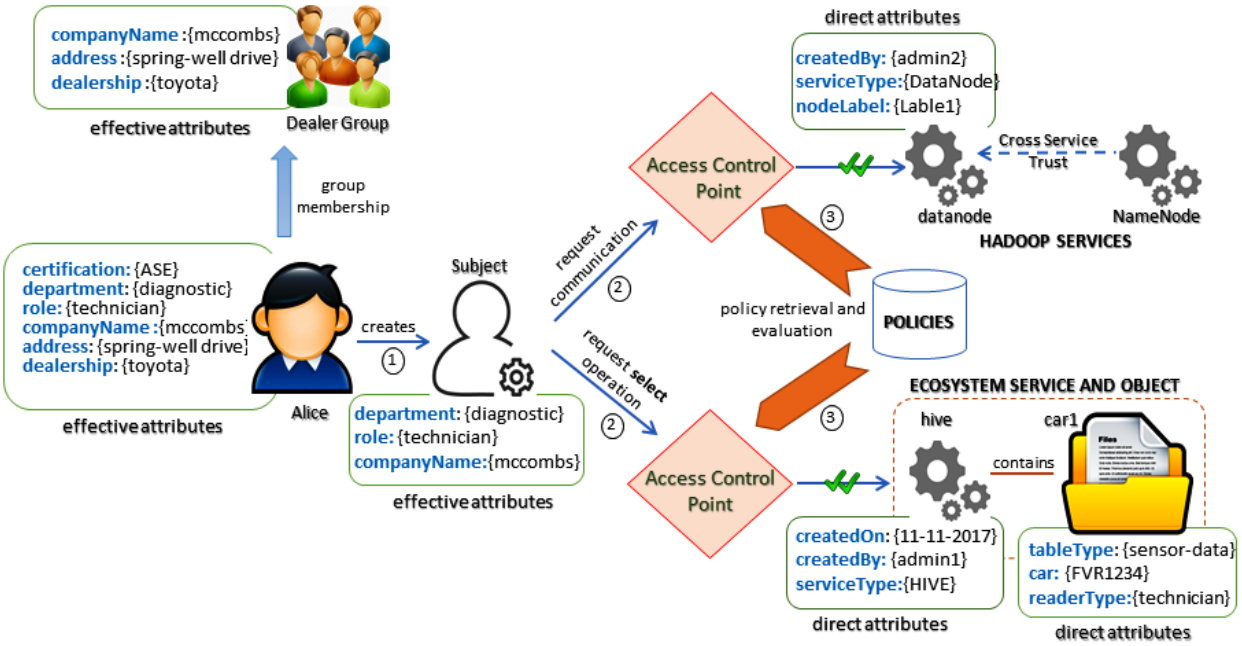


Figure 5.12: IoT Use-Case Illustrating ABAC Access Control in Hadoop Ecosystem

the customers. Let us suppose, a connected vehicle from car manufacturer Toyota is running on the road and is continuously generating data, which is stored in multi-tenant Hadoop data lake. This stored data can be used by various entities, including the car dealer or manufacturer for diagnostic services, by transportation or police department for over-speeding check, by insurance company to understand driver driving behaviour or by a doctor who is continuously monitoring the heart-rate of the patient driver. Each of these users must have different levels of access to data in the Hadoop data lake and are only required to know what they should need to know to perform their functions following the principle of least privileges, and without compromising the integrity and privacy of data. Further, for analysis purposes these users will be also running jobs or applications in the Hadoop cluster, which must cater the needs of all users without unwarranted resource constraints.

Figure 5.12 illustrates the use cases to reflect the importance of attribute based system in such distributed and multi-tenant environment like Hadoop. In this case, a user Alice is assigned to a Dealer group, which makes it inherit the attributes of Dealer group, yielding the effective attributes for Alice. Here, the attributes of Dealer group `(companyName: mccombs, address: spring-well drive, dealership: Toyota)` are added to the di-

rect attributes of Alice user (certification: ASE, department: diagnostic, role: technician). The benefits of user to group assignment are evident, since with single administrative operation all the attributes of Dealer group are assigned to Alice. Further, when Alice creates a subject, the subject inherits subset of the effective attributes of Alice. Also, other entities shown such as Hadoop services (datanode, NameNode), Ecosystem service (hive) and object (Table car1) in the system, are also assigned direct attributes by a security administrator. Security policies are defined by the architect and stored in the central policy server. Cross Hadoop services trust relation is also established which will be discussed more in the following part of this section. The numbers in the figure define the sequence of access control process where the subject is first created, which initiates requests to perform different operations on objects and services. These requests are intercepted by the access control decision and enforcement point (shown as rhombus), which will retrieve polices from the central server to make an access control decision.

Let us assume that the following security policy (referred as policy 1) is created by an administrator in the system to control access to some Ecosystem service:

$$\text{Authorization}_{\text{access}}(\text{s:S, es:ES}) \equiv$$
$$\text{diagnostic} \in \text{effective}_{\text{department}}(\text{s}) \wedge \text{technician} \in$$
$$\text{effective}_{\text{role}}(\text{s}) \wedge \text{serviceType(es)} = \text{HIVE} \wedge \text{createdBy(es)} =$$
$$\text{admin1}$$

This policy states that a user (or subject) belonging to diagnostic department with technician role can access ecosystem service of type HIVE which was created by admin1. Clearly, if subject s : S created by a user and an ecosystem service es : ES satisfy the stated policy condition (i.e. evaluates to True), then access operation will be granted on service es to subject s. This policy can be enforced by Apache Knox [6], which offers a single point gateway to multiple services inside Hadoop ecosystem. Another security policy (referred as policy 2) is created to determine if select operation is allowed by a subject s : S on an object ob : OB in ecosystem service es : ES:

$$\text{Authorization}_{\text{select}}(\text{s:S, es:ES, ob:OB}) \equiv$$
$$\text{Authorization}_{\text{access}}(\text{s:S, es:ES}) = \text{True} \wedge \text{diagnostic} \in$$

$$\texttt{effective}_{\texttt{department}}\texttt{(s)} \wedge \texttt{effective}_{\texttt{role}}\texttt{(s)} \in \texttt{readerType(ob)} \wedge$$
$$\texttt{tableType(ob) = sensor-data} \wedge \texttt{car(ob) = FVR1234}$$

This policy requires a subject to perform select operation on an object ob belonging to Ecosystem service es if the user belongs to diagnostic department and the effective roles of the user belong to readerType attribute of the object, tableType attribute of object having value sensor-data and car attribute with value FVR1234. It should be noted that, this authorization function has a condition $\texttt{Authorization}_{\texttt{access}}\texttt{(s:S, es:ES)}$ = True, stating that subject s must be first allowed to access ecosystem service es before its underlying object ob is allowed to be operated by subject s.

Let us say, in our use-case a user Alice from a car dealer wants to read the data of a car which is stored in Hadoop data lake. As a security requirement, Alice can only access data through Apache Hive ecosystem service with no direct access to data at HDFS level. Alice has some attributes which are its own, but some are also inherited from the car dealer as being a part of its employee. For this access to authorize, Alice must first be allowed to access Apache Hive ecosystem service and then allowed to read the table inside it. Looking at the effective attributes of user Alice, the subject created by Alice and the attributes of service hive and object table car1, it can be well understood that the policy 1 and policy 2 are satisfied by subject of user Alice. Therefore, select operation by Alice on table car1 is allowed by the defined security policies. Let us suppose another user Bob from the same car dealer but in a different department (say sales) tries to perform select operation on the same object table car1. The operation will not be allowed as the value for department attribute for Bob will be sales, which will not satisfy the above stated policies. Similar set of policies can be defined in the system to cater various other use-cases and security requirements in Hadoop data lake. For example, some user may only be allowed to access HDFS files directly without access through Apache Hive, or some may be allowed to submit YARN applications only. Policies can also be defined to prevent denial of resource attacks where specific users are only allowed to submit jobs to YARN capacity or fair scheduler queues which have limited set of resources attached to them. A sample security policy to restrict submitting YARN applications to only specific users can be defined for YARN capacity scheduler queues as:

$$\texttt{Authorization}_{\texttt{submit}}\texttt{(s:S, es:ES, ob:OB)} \equiv$$
$$\texttt{Authorization}_{\texttt{access}}\texttt{(s:S, es:ES) = True} \wedge \texttt{diagnostic} \in$$
$$\texttt{effective}_{\texttt{department}}\texttt{(s)} \wedge \texttt{technician} = \texttt{effective}_{\texttt{role}}\texttt{(s)} \wedge$$
$$\texttt{queueType(ob) = dept-diagnostic} \wedge \texttt{queueAdmin(ob) = admin2}$$

In this case, $\texttt{Authorization}_{\texttt{access}}\texttt{(s:S, es:ES)}$ = True signifies that a user with certain attributes must be allowed to access YARN ResourceManager (approved by a separate policy) before allowed to submit applications to its queues which have attributes queuetype having value dept-diagnostic and queueAdmin with value admin2.

Similar attribute based access control policies can be also defined for controlling user access to Hadoop daemon services like HDFS DataNode, NameNode etc. In Figure 5.12, we created a Hadoop service DataNode 'datanode' which has a set of attributes directly assigned to it. An access control list (ACL) `security.client.datanode.protocol.acl` is currently defined in Hadoop to control the communication between user clients and DataNodes to retrieve data blocks. The following security policy can be used in place of ACL to control this access:

$$\texttt{Authorization}_{\texttt{access}}\texttt{(s:S, hs:HS)} \equiv$$
$$\texttt{diagnostic} \in \texttt{effective}_{\texttt{department}}\texttt{(s)} \wedge \texttt{technician} \in$$
$$\texttt{effective}_{\texttt{role}}\texttt{(s)} \wedge \texttt{serviceType(hs) = DataNode} \wedge \texttt{createdBy(hs)}$$
$$\texttt{= admin2}$$

Clearly, if user Alice subject tries to access DataNode service 'datanode', the aforementioned policy will allow Alice to access service 'datanode' which will serve the purpose of defining `security.client.datanode.protocol.acl` ACL. Similar attribute based policies can be stated for other service level authorization ACLs also.

Another Hadoop service NameNode has also been shown in Figure 5.12 which trusts DataNode service 'datanode'. Cross Hadoop service trust relation is needed to control cross Hadoop services communication which is currently controlled by ACLs. For example, `security.datanode.protocol.acl` controls communication between DataNode and NameNode. These can be replaced by defining trust relations between Hadoop services. As shown

in Figure 5.12, DataNode 'datanode' has a cross service trust with NameNode, meaning datanode can access NameNode where NameNode is trustor and 'datanode' is trustee service. To state this requirement, we assume to have a cross Hadoop service Type-α trust type, written as NameNode $\trianglelefteq_\alpha$ datanode, where the trust is initiated by NameNode and service access to NameNode is also controlled by NameNode service. Other trust types can also be considered depending on use-cases requirements but for simplicity, we restrict to Type-α trust type in HeABAC model.

As noted, these real world use-cases illustrate how attribute-based access controls can be enforced into this dynamic and distributed environment, where users have different access needs. Fine grained requirements of a multi-tenant Hadoop data lake, where a user can access one service but not other, or two users having different levels of access to the same object can be truly catered by this HeABAC authorization model. Further, the use of trust in cross Hadoop service communication obviates the need to define service level authorization ACLs in the system.

CHAPTER 6: CONCLUSION

This chapter summarizes the contributions of this dissertation and provide some future research directions to solve open problems we came across during our work.

6.1 Summary

The work discussed in this dissertation contributes to the fundamental aspects of attribute based access control (ABAC) and extends its application to current technology domains and real world scenarios including cloud assisted Smart Cars and Big Data.

First, we propose a generalized URA97 model administration for Hierarchical Group and Attribute Based Access Control (HGABAC), called GURA_G. Propositional logic conditions together with administrative roles are defined to make administrative decisions for assignment of attributes and groups to users. GURA_G consists of three sub-models: User Attribute Assignment (UAA), User-Group Attribute Assignment (UGAA) and User to User-Group Assignment (UGA). The authorization relations in UAA and UGAA control addition and deletion of direct attributes from users and user-groups respectively. UGA governs assignment and removal of a user from user-groups based on the current membership (or non-membership) and attributes of the user. Next, we discuss the reachability problem to understand what attributes a user can attain based on set of administrative rules created using GURA_G. We defined a restricted form of GURA_G, referred as rGURA_G and classified three schemes $\text{rGURA}_{\text{G}_0}$, $\text{rGURA}_{\text{G}_1}$ and $\text{rGURA}_{\text{G}_{1+}}$ to discuss different reachability solutions. In general, we proved that the reachability problem for rGURA_G scheme is intractable as PSPACE-complete but with certain restrictions the problem is tractable for which we discussed polynomial time algorithms.

Second, we proposed an authorization framework for cloud assisted smart cars and vehicular IoT. The research provides security requirements and discusses several access control decision and enforcement points necessary in the dynamic ecosystem of connected vehicles. We presented an Extended Access Control Oriented (E-ACO) architecture which introduces the novel concept of

clustered objects (objects with several smart sensors and applications) like cars, traffic infrastructures or smart homes. The architecture envisions IoV to have both fog and cloud instances where fog can be static or dynamically built using vehicular cloud or fixed roadside base stations. Authorization framework discusses different communication and data exchange scenarios along with access control approaches in E-ACO layers. Real-world use-cases in single and multi-cloud scenarios with access control requirements justifies the need and use of authorization framework for vehicular IoT. Further in this chapter, we presented a fine-grained attribute-based access control model for real-time and location-centric smart cars ecosystem. The model (referred as CV-ABAC$_{\mathrm{G}}$) introduces the novel notion of dynamic groups in relation to connected vehicles and emphasizes its relevance in this context. Besides considering system wide authorization policies, this model also supports personal preference policies for different users, which is required in today's privacy conscious world. Several real world use-cases are discussed and a proof of concept implementation of our CV-ABAC$_{\mathrm{G}}$ model is shown in AWS. This implementation demonstrates how moving vehicles can be dynamically assigned to location groups and sub-groups defined in the system based on the current GPS coordinates and vehicle-type, which is an important contribution. Performance has been evaluated against time taken to determine activity access control decision when groups and ABAC policies are used against when no policies are available.

Finally, we proposed access control models and extensions for most widely used Big Data platform, called Hadoop. We presented the first formalized access control model called HeAC for Hadoop ecosystem. Besides the regular permissions including objects and operations, this model also includes object attribute values (represented as tags) in object permissions. We extended HeAC model to propose Object-Tagged RBAC model (OT-RBAC) which preserves role based permission assignment and presents a novel approach for adding object attributes to RBAC. We proposed an implementation in open-source Apache Ranger using context enricher and condition evaluators along with performance comparison graph. Further, we proposed an attribute-based access control model for Hadoop ecosystem, referred to as HeABAC, which offers fine grained and flexible authorization policies required in multi-tenant distributed system like Hadoop.

6.2 Future Work

Some potential future research directions which can be explored are as follows:

- **ABAC Administrative Models and Reachability:** As GURA_G proposes manual assignment of attribute values and user-groups to users, a potential research thrust can be to develop automated GURAG like model. An administrative model for group hierarchies and objects can also be a future prospect. Also instead of using the role of administrator, if user attributes can be used to define the administrative privileges, can also be explored. For future works, we can develop additional polynomial algorithms for some restricted forms and perform reachability analysis on other types of queries like effective user groups or minimum number of administrative requests to satisfy reachability query.

- **Smart Cars and Intelligent Transportation:** An important direction in cloud assisted smart cars is to extend CV-ABAC_G model to introduce in-vehicle security and built risk aware trust-based models for smart vehicles ecosystem. Further, location privacy preserving approaches such as homomorphic encryption and other anonymity techniques can be used to complement and extend CV-ABAC_G which can mitigate location sharing concerns without effecting its advantages and application. Multi-tenant and federation supported cloud or fog architectures can be proposed to ensure cross tenant trust. V2X trusted communication and privacy concerns also need further investigation, which can use edge cloudlets to ensure real time trusted communication and messages exchange in intelligent transportation.

- **Big Data and Hadoop Ecosystem:** Different Big Data platforms including NoSQL data stores like MongoDB must be studied and fine grained access control solutions can be proposed. Some privacy preserving policies and practices reflected using content based and purpose based access control need thorough investigation. Also, since the Hadoop data lake is used by multiple tenants it would be interesting to introduce data ingestion security into the system to secure data at HDFS data nodes level.

BIBLIOGRAPHY

[1] Amazon Web Services (AWS) - Cloud Computing Services. https://aws.amazon.com. [Accessed: 2016-11-08].

[2] Apache Ambari. http://ambari.apache.org/. [Accessed: 2016-11-14].

[3] Apache Hadoop. http://hadoop.apache.org/. [Accessed: 2017-05-17].

[4] Apache HBase. http://hbase.apache.org/. [Accessed: 2016-12-08].

[5] Apache Hive. http://hive.apache.org/. [Accessed: 2016-12-05].

[6] Apache Knox. https://knox.apache.org/. [Accessed: 2018-02-03].

[7] Apache Ranger. http://ranger.apache.org/. [Accessed: 2016-12-03].

[8] Apache Sentry. https://sentry.apache.org/. [Accessed: 2017-01-10].

[9] Apache Storm. http://storm.apache.org/. [Accessed: 2017-01-03].

[10] Cloudera. http://www.cloudera.com/products/apache-hadoop.html. [Accessed: 2017-05-13].

[11] Data Age 2025: The Evolution of Data to Life-Critical. https://www.idc.com/. [Accessed: 2017-02-03].

[12] Every Day Big Data Statistics 2.5 Quintillion Bytes Of Data Created Daily. http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data/-created-daily. [Accessed: 2018-02-03].

[13] Hortonworks. https://www.hortonworks.com/. [Accessed: 2017-03-13].

[14] Microsoft Azure. https://azure.microsoft.com. [Accessed: 2016-11-10].

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

[16] Connected vehicles and your privacy. https://www.its.dot.gov/factsheets/pdf/Privacy_factsheet.pdf, 2014. [Accessed: 2018-02-13].

[17] Building Autonomous and Connected Vehicle Systems with the Vortex IoT Data Sharing Platform. *Prismtech*, 2015.

[18] Big Data: Securing Intel IT's Apache Hadoop Platform. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/big-data-securing-intel-it-apache-hadoop-platform-paper.pdf, 2016. [Accessed: 2017-01-16].

[19] Convergence of secure vehicular ad-hoc network and cloud in internet of things. http://mahbubulalam.com/convergence-of-secure-vehicular-ad-hoc-network-and-cloud-in-iot/, 2016. [Accessed: 2018-02-01].

[20] Securing Hadoop: Security Recommendations for Hadoop Environments. https://securosis.com/assets/library/reports/Securing_Hadoop_Final_V2.pdf, 2016. [Accessed: 2018-02-03].

[21] 2017 Data Breaches - The worst So Far. https://www.identityforce.com/blog/2017-data-breaches, 2017. [Accessed: 2017-01-13].

[22] 2017 Roundup Of Internet Of Things Forecasts. https://www.forbes.com/sites/louiscolumbus/2017/12/10/2017-roundup-of-internet-of-things-forecasts/#67005b6a1480, 2017. [Accessed: 2018-05-03].

[23] Connected car. https://en.wikipedia.org/wiki/Connected_car, 2017. [Accessed: 2018-01-13].

[24] Securing The Connected Vehicle. *Thales E-Security*, 2017.

[25] AWS Greengrass. https://aws.amazon.com/greengrass/, 2018. [Accessed: 2018-05-27].

[26] AWS IoT. https://aws.amazon.com/iot/, 2018. [Accessed: 2018-05-09].

[27] AWS Lambda. https://aws.amazon.com/lambda/, 2018. [Accessed: 2018-05-20].

[28] AWS SDK for Python (Boto3). https://aws.amazon.com/sdk-for-python/, 2018. [Accessed: 2018-05-23].

[29] AWS Simple Notification Service. https://aws.amazon.com/sns/, 2018. [Accessed: 2018-05-20].

[30] Cloud IoT Core. https://cloud.google.com/iot-core/, 2018. [Accessed: 2018-02-03].

[31] Dedicated Short Range Communications. https://en.wikipedia.org/wiki/Dedicated_short-range_communications, 2018. [Accessed: 2018-08-07].

[32] Device Twins. https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-device-twins, 2018. [Accessed: 2018-02-03].

[33] Google Maps Platform. https://cloud.google.com/maps-platform/, 2018. [Accessed: 2018-05-09].

[34] M. Aazam and et al. Cloud of Things: Integrating Internet of Things and cloud computing and the issues involved. In *Proceedings of IBCAST*, pages 414–419, 2014.

[35] A. Al-Fuqaha and et al. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Comm. Surveys & Tutorials*, pages 2347–2376, 2015.

[36] Mohammad A Al-Kahtani and Ravi Sandhu. A model for attribute-based user-role assignment. In *Proceedings of IEEE Annual Computer Security Applications Conference*, pages 353–362, 2002.

[37] Asma Alshehri and Ravi Sandhu. Access control models for cloud-enabled internet of things: A proposed architecture and research agenda. In *Proceedings of IEEE CIC*, pages 530–538, 2016.

[38] Asma Alshehri and Ravi Sandhu. Access Control Models for Virtual Object Communication in Cloud-Enabled IoT. In *Proceedings of IEEE IRI*, pages 16–25, 2017.

[39] Mikio Aoyama. Computing for the next-generation automobile. *IEEE Computer*, 45(6):32–37, 2012.

[40] Alessandro Armando, Michele Bezzi, Nadia Metoui, and Antonino Sabetta. Risk-based privacy-aware information disclosure. *International Journal of Secure Software Engineering (IJSSE)*, 6(2):70–89, 2015.

[41] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[42] Amazon AWS. Thing Shadows for AWS IoT. http://docs.aws.amazon.com/iot/latest/developerguide/iot-thing-shadows.html, 2017. [Accessed: 2018-01-25].

[43] Christer Bäckström and Bernhard Nebel. Complexity results for SAS+ planning. *Computational Intelligence*, 11(4):625–655, 1995.

[44] Nazia Badar, Jaideep Vaidya, Vijayalakshmi Atluri, and Basit Shafiq. Risk based access control using classification. In *Automated Security Management*, pages 79–95. Springer, 2013.

[45] Arosha K Bandara, Emil C Lupu, and Alessandra Russo. Using event calculus to formalise policy specification and analysis. In *Proceedings of IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 26–39, 2003.

[46] Jim Barbaresso and et al. USDOT's Intelligent Transportation Systems ITS Strategic Plan 2015- 2019. 2014.

[47] Mahmoud Barhamgi, Djamal Benslimane, Said Oulmakhzoune, Nora Cuppens-Boulahia, Frederic Cuppens, Michael Mrissa, and Hajer Taktak. Secure and privacy-preserving exe-

cution model for data services. In *Proceedings of International Conference on Advanced Information Systems Engineering*, pages 35–50. Springer, 2013.

[48] S. Bhatt, F. Patwa, and R. Sandhu. An access control framework for cloud-enabled wearable internet of things. In *Proceedings of IEEE CIC*, pages 328–338, 2017.

[49] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. Access control model for AWS internet of things. In *Proceedings of NSS*, pages 721–736. Springer, 2017.

[50] A. R. Biswas and R. Giaffreda. IoT and cloud convergence: Opportunities and challenges. In *Proceedings of IEEE WF-IoT*, pages 375–376, 2014.

[51] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog computing and its role in the internet of things. In *Proceedings of ACM MCC Workshop*, pages 13–16, 2012.

[52] A. Botta, W. de Donato, V. Persico, and A. Pescape. On the Integration of Cloud Computing and Internet of Things. In *Proceedings of IEEE FiCLOUD*, pages 23–30, 2014.

[53] Alessio Botta and et al. Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, pages 684 – 700, 2016.

[54] David Brown et al. Automotive Security Best Practice. *Intel Security*, 2015.

[55] V. G. Cerf. Access control and the internet of things. *IEEE Internet Computing*, 19(5):96–c3, Sept 2015.

[56] Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *Proceedings of IFIP Annual Conference on Data and Applications Security and Privacy*, pages 8–24. Springer, 2012.

[57] Laurence Cholvy and Frédéric Cuppens. Analyzing consistency of security policies. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 103–112, 1997.

[58] Lorenzo Cirio, Isabel F Cruz, and Roberto Tamassia. A role and attribute based access control system using semantic web technologies. In *Proceedings of OTM Confederated International Conferences on "On the Move to Meaningful Internet Systems"*, pages 1256–1266. Springer, 2007.

[59] Pietro Colombo and Elena Ferrari. Complementing MongoDB with advanced access control features: Concepts and research challenges. In *Proceedings of Italian Symposium on Advanced Database Systems*, pages 343–350, 2015.

[60] Pietro Colombo and Elena Ferrari. Privacy aware access control for Big Data: A research roadmap. *Big Data Research*, 2(4):145–154, 2015.

[61] J. Contreras, S. Zeadally, and J. A. Guerrero-Ibanez. Internet of vehicles: Architecture, protocols, and security. *IEEE Internet of Things*, pages 1–9, 2017.

[62] Michael J Covington and Manoj R Sastry. A contextual attribute-based access control model. In *Proceedings of OTM Confederated International Conferences on "On the Move to Meaningful Internet Systems"*, pages 1996–2006. Springer, 2006.

[63] Jason Crampton and George Loizou. Administrative scope: A foundation for role-based administrative models. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):201–231, 2003.

[64] Devaraj Das, Owen O'Malley, Sanjay Radia, and Kan Zhang. Adding security to Apache Hadoop. *Hortonworks, IBM*, 2011.

[65] M. DÃaz and et al. State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, pages 99 – 117, 2016.

[66] Philip Derbeko, Shlomi Dolev, Ehud Gudes, and Shantanu Sharma. Security and privacy aspects in MapReduce on clouds: A survey. *Computer Science Review*, 20:1–28, 2016.

[67] Sabrina De Capitani di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. Protecting access confidentiality with data distribution and swapping. In *Proceedings of IEEE International Conference on Big Data and Cloud Computing*, pages 167–174, 2014.

[68] A. Elmaghraby and M. Losavio. Cyber security challenges in smart cities: Safety, security and privacy. *Journal of advanced research*, 5(4):491–497, 2014.

[69] Mohamed Eltoweissy, Stephan Olariu, and Mohamed Younis. Towards autonomous vehicular clouds. *Ad hoc networks*, pages 1–16, 2010.

[70] ENISA. Cyber Security and Resilience of smart cars: Good practices and recommendations. https://www.enisa.europa.eu/publications/cyber-security-and-resilience-of-smart-cars, 2017. [Accessed: 2018-01-27].

[71] Y. Fangchun and et al. An overview of internet of vehicles. *China Communications*, 11(10):1–15, 2014.

[72] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3):224–274, 2001.

[73] Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of IEEE International Conference on Software Engineering*, pages 196–205, 2005.

[74] Keith Frikken, Mikhail Atallah, and Jiangtao Li. Attribute-based access control with hidden policies and hidden credentials. *IEEE Transactions on Computers*, 55(10):1259–1270, 2006.

[75] John Gantz et al. Digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2012.

[76] US GAO. Vehicle Cybersecurity . *GAO-16-350*, 2016, March.

[77] Gartner. Gartner Says By 2020, a Quarter Billion Connected Vehicles Will Enable New In-Vehicle Services and Automated Driving Capabilities. https://www.gartner.com/newsroom/id/2970017, 2015. [Accessed: 2017-10-25].

[78] Gartner. 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, 2017. [Accessed: 2018-01-13].

[79] M. Gerla. Vehicular cloud computing. In *Proceedings of IEEE Med-Hoc-Net*, 2012.

[80] M. Gerla, E. Lee, G. Pau, and U. Lee. Internet of vehicles: From intelligient grid to autonomous cars and vehicular clouds. In *Proceedings of IEEE WF-IoT*, pages 241–246, 2014.

[81] Matthew Gigli and Simon Koo. Internet of things: services and applications categorization. *Advances in Internet of Things*, 1(02):27, 2011.

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

[83] J. Gubbi and et al. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[84] Maanak Gupta, Farhan Patwa, James Benson, and Ravi Sandhu. Multi-layer authorization framework for a representative Hadoop ecosystem deployment. In *Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 183–190, 2017.

[85] Maanak Gupta, Farhan Patwa, and Ravi Sandhu. Object-Tagged RBAC Model for the Hadoop Ecosystem. In *Proceedings of IFIP Annual Conference on Data and Applications Security and Privacy*, pages 63–81. Springer, 2017.

[86] Maanak Gupta, Farhan Patwa, and Ravi Sandhu. POSTER: Access Control Model for the Hadoop Ecosystem. In *Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 125–127, 2017.

[87] Maanak Gupta, Farhan Patwa, and Ravi Sandhu. An Attribute-Based Access Control Model for Secure Big Data Processing in Hadoop Ecosystem. In *Proceedings of ACM Workshop on Attribute-Based Access Control*, pages 13–24, 2018.

[88] Maanak Gupta and Ravi Sandhu. The $\mathrm{GURA_G}$ Administrative Model for User and Group Attribute Assignment. In *Proceedings of International Conference on Network and System Security*, pages 318–332. Springer, 2016.

[89] Maanak Gupta and Ravi Sandhu. Authorization framework for secure cloud assisted connected cars and vehicular internet of things. In *Proceedings of ACM Symposium on Access Control Models and Technologies*, pages 193–204. ACM, 2018.

[90] Maanak Gupta and Ravi Sandhu. POSTER: Access Control Needs in Smart Cars. https://www.ieee-security.org/TC/SP2018/poster-abstracts/oakland2018-paper26-poster-abstract.pdf, 2018. [Accessed: 2018-10-04].

[91] Puneet Gupta, Scott D Stoller, and Zhongyuan Xu. Abductive analysis of administrative policies in rule-based access control. *IEEE Transactions on Dependable and Secure Computing*, 11(5):412–424, 2014.

[92] 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.

[93] Kevin Hamlen, Murat Kantarcioglu, Latifur Khan, and Bhavani Thuraisingham. Security issues for cloud computing. *Optimizing Information Security and Advancing Privacy Assurance: New Technologies*, 150, 2012.

[94] Michael A Harrison, Walter L Ruzzo, and Jeffrey D Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[95] J. Hernandez-Ramos and et al. Distributed capability-based access control for the internet of things. *Journal of Internet Services and Information Security*, 3(3/4):1–16, 2013.

[96] Andy Chunliang Hsu and Indrakshi Ray. Specification and enforcement of location-aware attribute-based access control for online social networks. In *Proceedings of ACM International Workshop on Attribute Based Access Control*, pages 25–34, 2016.

[97] 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.

[98] Vincent C Hu, Tim Grance, David F Ferraiolo, and D Rick Kuhn. An access control scheme for Big Data processing. In *Proceedings of IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom)*, pages 1–7, 2014.

[99] Vincent C Hu, D Richard Kuhn, and David F Ferraiolo. Attribute-based access control. *IEEE Computer*, (2):85–88, 2015.

[100] Jean-Pierre Hubaux, Srdjan Capkun, and Jun Luo. The security and privacy of smart vehicles. *IEEE Security & Privacy*, 2(3):49–55, 2004.

[101] Rasheed Hussain and et al. Rethinking vehicular communications: Merging VANET with cloud computing. In *Proceedings of IEEE CloudCom*, pages 606–609, 2012.

[102] Trent Jaeger, Xiaolan Zhang, and Antony Edwards. Policy management using access control spaces. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):327–364, 2003.

[103] Sushil Jajodia, Pierangela Samarati, and VS Subrahmanian. A logical language for expressing authorizations. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 31–42. IEEE, 1997.

[104] Somesh Jha, Ninghui Li, Mahesh Tripunitara, Qihua Wang, and William Winsborough. Towards formal verification of role-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 5(4):242–255, 2008.

[105] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Proceedings of IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer, 2012.

[106] Xin Jin, Ram Krishnan, and Ravi Sandhu. A role-based administration model for attributes. In *Proceedings of ACM International Workshop on Secure and Resilient Architectures and Systems*, pages 7–12, 2012.

[107] Xin Jin, Ram Krishnan, and Ravi Sandhu. Reachability analysis for role-based administration of attributes. In *Proceedings of ACM workshop on Digital identity management*, pages 73–84, 2013.

[108] Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: role-centric attribute-based access control. In *Proceedings of International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 84–96. Springer, 2012.

[109] James BD Joshi, Elisa Bertino, and Arif Ghafoor. An analysis of expressiveness and design issues for the generalized temporal role-based access control model. *IEEE Transactions on Dependable and Secure Computing*, 2(2):157–175, 2005.

[110] O. Kaiwartya and et al. Internet of vehicles: Motivation, layered architecture, network model, challenges, and future aspects. *IEEE Access*, 4:5356–5373, 2016.

[111] Sun Kaiwen and Yin Lihua. Attribute-role-based hybrid access control in the internet of things. In *Proceedings of APWeb*, pages 333–343. Springer, 2014.

[112] Evangelos A Kosmatos, Nikolaos D Tselikas, and Anthony C Boucouvalas. Integrating RFIDs and smart objects into a unified internet of things architecture. *Advances in Internet of Things*, 1(01):5, 2011.

[113] Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, and Artsiom Yautsiukhin. Towards attribute-based access control policy engineering using risk. In *Proceedings of International Workshop on Risk Assessment and Risk-driven Testing*, pages 80–90. Springer, 2013.

[114] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. Adding attributes to role-based access control. *Computer*, 43(6):79–81, 2010.

[115] Devdatta Kulkarni. A fine-grained access control model for key-value systems. In *Proceedings of ACM Conference on Data and application security and privacy*, pages 161–164, 2013.

[116] Swarun Kumar and et al. Carspeak: A content-centric network for autonomous driving. *SIGCOMM Computer Communication Review*, 42(4):259–270, August 2012.

[117] Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2):169, 2009.

[118] R. Lea and M. Blackstock. City Hub: A Cloud-Based IoT Platform for Smart Cities. In *Proceedings of IEEE CloudCom*, pages 799–804, Dec 2014.

[119] U. Lee and et al. Mobeyes: smart mobs for urban monitoring with a vehicular sensor network. *IEEE Wireless Communications*, pages 52–57, 2006.

[120] Ninghui Li, John C Mitchell, and William H Winsborough. Beyond proof-of-compliance: security analysis in trust management. *Journal of the ACM*, 52(3):474–514, 2005.

[121] Ninghui Li and Mahesh V Tripunitara. Security analysis in role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 9(4):391–420, 2006.

[122] Richard J Lipton and Lawrence Snyder. A linear time algorithm for deciding subject security. *Journal of the ACM*, 24(3):455–464, 1977.

[123] Haibing Lu, Yuan Hong, Yanjiang Yang, Lian Duan, and Nazia Badar. Towards user-oriented RBAC model. *Journal of Computer Security*, 23(1):107–129, 2015.

[124] Rongxing Lu, Hui Zhu, Ximeng Liu, Joseph K Liu, and Jun Shao. Toward efficient and privacy-preserving computing in big data era. *IEEE Network*, 28(4):46–50, 2014.

[125] Francisco Moyano, Carmen Fernandez-Gago, and Javier Lopez. A conceptual framework for trust models. In *Proceedings of International Conference on Trust, Privacy and Security in Digital Business*, pages 93–104. Springer, 2012.

[126] NHTSA. NHTSA and Vehicle CyberSecurity. *NHTSA Report*, 2016.

[127] NHTSA. Cybersecurity Best Practices for Modern Vehicles. *NHTSA Report No. DOT HS 812 333*, 2016, October.

[128] NIST. Framework for Cyber-Physical Systems. https://www.nist.gov/itl/applied-cybersecurity/nist-initiatives-iot, 2016. [Accessed: 2018-01-13].

[129] M. Nitti and et al. The virtual object as a major element of the internet of things: a survey. *IEEE Comm. Surveys & Tutorials*, pages 1228–1240, 2016.

[130] David Nunez, Isaac Agudo, and Javier Lopez. Delegated access for Hadoop clusters in the cloud. In *Proceedings of IEEE International Conference on Cloud Computing Technology and Science*, pages 374–379, 2014.

[131] Stephan Olariu and et al. Taking VANET to the clouds. *International Journal of Pervasive Computing and Communications*, 7(1):7–21, 2011.

[132] Owen O'Malley, Kan Zhang, Sanjay Radia, Ram Marti, and Christopher Harrell. Hadoop security design. *Yahoo, Inc., Tech. Rep*, 2009.

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

[134] Aafaf Ouaddah and et al. Access control in the internet of things: Big challenges and new opportunities. *Computer Networks*, 112:237–262, 2017.

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

[136] Christopher Poulen. Driving security: Cyber assurance for next-generation vehicles. *IBM Global Business Services*, 2014.

[137] Navid Pustchi, Ram Krishnan, and Ravi Sandhu. Authorization federation in IaaS multi cloud. In *Proceddings of ACM International Workshop on Security in Cloud Computing*, pages 63–71, 2015.

[138] PV Rajkumar and Ravi Sandhu. Safety decidability for pre-authorization usage control with finite attribute domains. *IEEE Transactions on Dependable and Secure Computing*, 13(5):582–590, 2016.

[139] Brian Russell and et al. Observations and Recommendations on Connected Vehicle Security. *Cloud Security Alliance*, 2017.

[140] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. The ARBAC97 model for role-based administration of roles. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):105–135, 1999.

[141] Ravi S Sandhu. The schematic protection model: its definition and analysis for acyclic attenuating schemes. *Journal of the ACM*, 35(2):404–432, 1988.

[142] Ravi S Sandhu. The typed access matrix model. In *Proceedings of IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136, 1992.

[143] Ravi S Sandhu. Lattice-based access control models. *Computer*, (11):9–19, 1993.

[144] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

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

[146] Johannes Sänger, Christian Richthammer, Sabri Hassan, and Günther Pernul. Trust and big data: A roadmap for research. In *Proceedings of IEEE International Workshop on Database and Expert Systems Applications*, pages 278–282, 2014.

[147] Chayan Sarkar and et al. Diat: A scalable distributed architecture for IoT. *IEEE Internet of Things journal*, 2(3):230–239, 2015.

[148] Amit Sasturkar, Ping Yang, Scott D Stoller, and CR Ramakrishnan. Policy analysis for administrative role based access control. In *Proceedings of IEEE Computer Security Foundations Workshop*, page 13, 2006.

[149] Walter J Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

[150] Andreas Schaad and Jonathan D Moffett. A lightweight approach to specification and analysis of role-based access control extensions. In *Proceedings of ACM symposium on Access control models and technologies*, pages 13–22, 2002.

[151] Ludwig Seitz, Göran Selander, and Christian Gehrmann. Authorization framework for the internet-of-things. In *Proceedings of IEEE WoWMoM*, pages 1–6, 2013.

[152] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K Rajamani, Janice Tsai, and Jeannette M Wing. Bootstrapping privacy compliance in big data systems. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 327–342, 2014.

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

[154] Priya P Sharma and Chandrakant P Navdeti. Securing Big Data Hadoop: A review of security issues, threats and solution. *International Journal Computer Science Information Technology*, 5, 2014.

[155] Hai-bo Shen and Fan Hong. An attribute-based access control model for web services. In *Proceedings of IEEE International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 74–79, 2006.

[156] Jordi Soria-Comas and Josep Domingo-Ferrer. Big data privacy: challenges to privacy principles and models. *Data Science and Engineering*, 1(1):21–28, 2016.

[157] Scott D Stoller, Ping Yang, Mikhail I Gofman, and CR Ramakrishnan. Symbolic reachability analysis for parameterized administrative role-based access control. *Computers & Security*, 30(2):148–164, 2011.

[158] Scott D Stoller, Ping Yang, C R Ramakrishnan, and Mikhail I Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of ACM Conference on Computer and communications security*, pages 445–455. ACM, 2007.

[159] Yunchuan Sun and et al. Security and privacy in the internet of vehicles. In *Proceedings of IEEE IIKI*, pages 116–121, 2015.

[160] Bo Tang and Ravi Sandhu. Cross-tenant trust models in cloud computing. In *Proceddings of IEEE International Conference on Information Reuse and Integration (IRI)*, pages 129–136, 2013.

[161] Bo Tang and Ravi Sandhu. Extending openstack access control with domain trust. In *Proceddings of International Conference on Network and System Security*, pages 54–69. Springer, 2014.

[162] Bo Tang, Ravi Sandhu, and Qi Li. Multi-tenancy authorization models for collaborative cloud services. *Concurrency and Computation: Practice and Experience*, 27(11):2851–2868, 2015.

[163] Omer Tene and Jules Polonetsky. Big data for all: Privacy and user control in the age of analytics. *Nw. J. Tech. & Intell. Prop.*, 11:xxvii, 2012.

[164] Omer Tene and Jules Polonetsky. Privacy in the age of big data: a time for big decisions. *Stanford Law Review Online*, 64:63, 2012.

[165] Toyota. Toyota-to-launch-smartphone-on-wheels. https://www.2wglobal.com/news-and-insights/articles/features/Toyota-to-launch-smartphone-on-wheels/, 2011. [Accessed: 2018-02-03].

[166] Mahesh V Tripunitara and Ninghui Li. A theory for comparing the expressive power of access control models. *Journal of Computer Security*, 15(2):231–272, 2007.

[167] Mahesh V. Tripunitara and Ninghui Li. The foundational work of Harrison-Ruzzo-Ullman Revisited. *IEEE Transactions on Dependable and Secure Computing*, 10(1):28–39, 2013.

[168] Huseyin Ulusoy, Pietro Colombo, Elena Ferrari, Murat Kantarcioglu, and Erman Pattuk. GuardMR: Fine-grained security policy enforcement for MapReduce systems. In *Proceedings of ACM Symposium on Information, Computer and Communications Security*, pages 285–296, 2015.

[169] Huseyin Ulusoy, Murat Kantarcioglu, Erman Pattuk, and Kevin Hamlen. Vigiles: Fine-grained access control for mapreduce systems. In *Proceedings of IEEE International Congress on Big Data*, pages 40–47, 2014.

[170] European Union. Certificate Policy for Deployment and Operation of European Cooperative Intelligent Transport Systems (C-ITS). https://ec.europa.eu/transport/sites/transport/files/c-its_certificate_policy_release_1.pdf, 2017. [Accessed: 2018-01-27].

[171] European Union. Security Policy & Governance Framework for Deployment and Operation of European Cooperative Intelligent Transport Systems (C-ITS). https://ec.europa.eu/transport/sites/transport/files/c-its_security_policy_release_1.pdf, 2017. [Accessed: 2018-01-27].

[172] USAToday. Chinese group hacks a Tesla for the second year in a row. https://www.usatoday.com/story/tech/2017/07/28/chinese-group-hacks-tesla-second-year-row/518430001/, 2017. [Accessed: 2017-12-03].

[173] USDOT. Connected Vehicles and Your Privacy. https://www.its.dot.gov/factsheets/pdf/Privacy_factsheet.pdf, 2016. [Accessed: 2018-02-17].

[174] USDOT. Securty Credential Management System. https://www.its.dot.gov/resources/scms.htm, 2016. [Accessed: 2018-01-13].

[175] Timo van Roermund. Secure Connected Cars for a Smarter World. *NXP Semiconductors*, 2015.

[176] Sabrina De Capitani Di Vimercati, Sara Foresti, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. Shuffle index: efficient and private access to outsourced data. *ACM Transactions on Storage (TOS)*, 11(4):19, 2015.

[177] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Proceedings of IEEE INFOCOM*, pages 1–9, 2010.

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

[179] Evan Welbourne and et al. Building the internet of things using RFID: the RFID ecosystem experience. *IEEE Internet computing*, 13(3), 2009.

[180] Md Whaiduzzaman and et al. A survey on vehicular cloud computing. *Journal of Network and Computer Applications*, 40:325–344, 2014.

[181] Tom White. *Hadoop: The definitive guide*. " O'Reilly Media, Inc.", 2012.

[182] Wired. *Hackers Remotely Kill a Jeep on the Highway-With Me in It*, 2015. [Accessed: 2018-08-07].

[183] Konrad Wrona, Sander Oudkerk, Alessandro Armando, Silvio Ranise, Riccardo Traverso, Lisa Ferrari, and Richard McEvoy. Assisted content-based labelling and classification of documents. In *Proceedings of IEEE International Conference on Military Communications and Information Systems (ICMCIS)*, pages 1–7, 2016.

[184] Ning Ye and et al. An efficient authentication and access control scheme for perception layer of internet of things. *Applied Mathematics and Information Sciences*, 8(4):1617–1624, 2014.

[185] Jianming Yong, Elisa Bertino, and Mark Toleman Dave Roberts. Extended RBAC with role attributes. In *Proceedings of Pacific Asia Conference on Information Systems*, page 8, 2006.

[186] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of ACM Symposium on Information, Computer and Communications Security*, pages 261–270, New York, NY, USA, 2010.

[187] Eric Yuan and Jin Tong. Attributed based access control (ABAC) for web services. In *Proceedings of IEEE International Conference on Web Services*, 2005.

[188] Guoping Zhang and Jiazheng Tian. An extended role based access control model for the internet of things. In *Proceedings of IEEE ICINA*, pages 319–323, 2010.

[189] Jiaqi Zhao, Lizhe Wang, Jie Tao, Jinjun Chen, Weiye Sun, Rajiv Ranjan, Joanna Kołodziej, Achim Streit, and Dimitrios Georgakopoulos. A security framework in G-Hadoop for big data computing across distributed cloud data centres. *Journal of Computer and System Sciences*, 80(5):994–1007, 2014.

VITA

Maanak Gupta was born and brought up in Karnal, Haryana, India. He completed his Bachelor of Technology (B.Tech) degree in Computer Engineering from Kurukshetra University in year 2010. Subsequently, he joined Master of Science (M.S.) program in Information Systems at Northeastern University, Boston, USA and graduated in 2012. After graduation, he worked as assistant professor for around two years at Dehradun Institute of Technology (DIT) University, Dehradun, India in the Department of Computer Science. He enrolled in doctoral program at The University of Texas at San Antonio (UTSA) in Fall 2014, and later joined as research assistant at the Institute for Cyber Security under Professor Ravi Sandhu. He also received M.S. in Computer Science at UTSA with specialization in Computer and Information Security. His primary area of research includes security and privacy in cyber space. In particular, he is interested in studying foundational aspects of access control and their applications in real world 21st century technologies including Smart Cars, Internet of Things, Cloud Computing, Blockchain and Big Data.