POLICY REVIEW IN ATTRIBUTE-BASED ACCESS CONTROL

by

SHERIFDEEN LAWAL, M.Sc.

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

DOCTOR OF PHILOSOPHY IN ELECTRICAL ENGINEERING

COMMITTEE MEMBERS:
Ram Krishnan, Ph.D., Chair
Guenevere Chen, Ph.D.
Maribel Fernandez, Ph.D.
Eugene John, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Electrical and Computer Engineering
December 2021

DEDICATION

To my family

ACKNOWLEDGEMENTS

I cannot overstate my heartfelt gratitude to Professor Dr. Ram Krishnan. It is a great honor for me to complete my dissertation under his tutelage. I have learned from him not to give up and exercise patience in solving technical problems. Your approach to mentorship is exemplary.

It is my great pleasure to extend my gratitude towards Professor Dr. Chen Guenevere for motivation and constructive criticism of my dissertation. I like to make a public show of my gratitude towards Professor Dr. Eugene John. Thank you for always making yourself available to me, constructive feedback on my dissertation, expert advice, and encouragement since my master's degree program. I like to express my gratitude to Professor Dr. Fernandez Maribel for her thought-provoking questions, valuable feedbacks, contribution to the subject of Attribute-Based Access Control and computer security at large.

I am thankful to our College of Engineering staff, Ms. LiPing Bein, and Department staff Ms. Khanh Nguyen for their around-the-clock help. A resounding thank you to the staff member of the Institute for Cyber Security, Ms. Suzanne Tanaka, for her swift and untiring assistance.

I am indebted to the unwavering support of the women in my life. To my mum Munirat, Iâm lost for words to express my gratitude for all you have sacrificed. To my wife Sekinat, your enduring love, patience, and perseverance shall continue to pay off. To my two sisters, Khadijat and Tawakalt, all your rescue missions linger. My gracious and amusing daughters Aisha and Alimah, thank you for lightening my world. A special thank you to my brothers, Ismail, Ahmed, and Akeem, for rallying round me. I am thankful to uncles Ibrahim and Momoh-Jimoh for making America feels like home and taking care of my needs.

This work is partially supported by NSF grants HRD-1736209 and CNS-1553696

December 2021

POLICY REVIEW IN ATTRIBUTE-BASED ACCESS CONTROL

SHERIFDEEN LAWAL, Ph.D.
The University of Texas at San Antonio, 2021

Supervising Professors: Ram Krishnan, Ph.D.

The Next Generation Access Control (NGAC), founded on the Policy Machine (PM), is a robust Attribute-Based Access Control (ABAC) framework that enables a structured and flexible approach for the establishment of the conventional access control models. The authorization state of the policy machine is an annotated Directed Acyclic Graph (DAG). Structurally, relations among attributes of the same type are hierarchical. This structure allows specifying authorization/revocation in multiple ways. However, one or more limitations can make most of the approaches to grant or revoke access inconsistent with existing policies. We proposed a variety of algorithms that provides the Policy Machine administrator a comprehensive list of all possible methods to authorize or revoke access using ABAC policy review. The approaches generated by these algorithms can help the PM administrator make an informed decision before access authorization or revocation.

This work began with a pilot study where we consider the policy review for authorization of an administrative access right, user assignment. The preliminary work evolved to a generic algorithm that reviews authorization policy for other administrative access rights. A thorough extension of the generic algorithm accommodates the policy review of authorization with constraints and revocation.

In recent times, as the number of blockchain use cases continues to grow, methods and technologies utilized by fraudsters continue to become sophisticated. The most notable form of cyber-attack utilizes a security breach in the internal security of blockchain systems, leading to illegal access to application services. A complex system like the blockchain network requires a dynamic, flexible, and scalable access control mechanism. There are numerous research efforts to leverage smart contracts in implementing access control based on blockchain. However, most of these contributions are either focused on blockchain-based access control for an off-chain resource. Other effects implement blockchain-based access control for a specific domain. This dissertation presents the first-ever implementation of the NIST NGAC (Policy Machine) system in a blockchain network. We utilized an instance of the Policy Machine for controlling access to assets in multiple blockchain ledgers. We implemented and evaluated the algorithms in this dissertation on the Hyperledger Fabric blockchain network.

TABLE OF CONTENTS

Acknowledgements iv

Abstract vi

List of Tables xii

List of Figures xiii

Chapter 1: Introduction & Motivation 1

1.1 The Need For Attribute-Based Access Control 3

1.2 Policy Reviews and Complexities of Attribute-Based Access Control 4

1.3 Problem Statement 5

1.4 Summary of Contributions 6

1.5 Organization of Dissertation 6

Chapter 2: Background And Literature Survey 7

2.1 Access Control Models 7

2.1.1 Discretionary Access Control (DAC) 7

2.1.2 Mandatory Access Control (MAC) 8

2.1.3 Role-Based Access Control (RBAC) 9

2.1.4 Attribute-Based Access Control (ABAC) 10

2.2 Policy Review in Access Control Models 11

2.2.1 Discretionary Access Control 11

2.2.2 Role-Based Access Control 12

2.2.3 Attribute-Based Access Control 12

2.3 Attribute Based Access Control 13

2.4 Attribute-Based Access Control Models 14

2.5 Policy Machine Overview 16

2.6 ABAC Policy Specification Methods 18

2.6.1 Enumerated Policy 18

2.6.2 Logical-Formula Policy 18

2.7 Administrative Role Based Access Control Policy Analysis 19

2.7.1 ARBAC97 19

2.7.2 Administrative TRBAC 20

2.8 Attribute Based Access Control in Hyperledger Fabric 21

Chapter 3: Administrative Access Authorization Through Policy Review 23

3.1 Preliminaries 23

3.1.1 Definitions 23

3.1.2 Assumption 24

3.1.3 Observations 25

3.1.4 Notations 25

3.1.5 Claim 26

3.1.6 Scope 26

3.2 Policy Review For Access Authorization 27

3.3 Illustrative Example 35

3.4 Algorithmic Design and Evaluation 37

3.4.1 Policy Review Algorithm for Generic Authorization 37

3.4.2 Performance Evaluation of Access Authorization Review Using Networkx 39

Chapter 4: Constrained Access Authorization And Revocation Through Policy Review 43

4.1 Policy Review For Constrained Access Authorization 43

4.1.1 Derived functions 44

4.1.2 Constrained Access Authorization Methodologies 45

4.1.3 Constrained Access Enablers 48

4.2 Policy Review For Access Revocation 49
4.3 Illustrative Example 50
4.3.1 Policy Review for Scenario-I 50
4.3.2 Policy Review for Scenario-II 53
4.4 Algorithmic Design and Evaluation 54
4.4.1 Policy Review Algorithms for Constrained Authorization and Revocation . 54
4.4.2 Performance Evaluation of Constrained Access Authorization and Revocation in Hyperledger Fabric 59

Chapter 5: Implementation of ABAC Review In Hyperledger Fabric Blockchain 62
5.1 Background 62
5.2 Hyperledger Fabric Overview 63
5.3 Hyperledger Fabric Blockchain Network Development 67
5.3.1 Tools and Components 67
5.3.2 Network Initialization 68
5.3.3 Transaction flow 68
5.4 Access control in HLF 69
5.4.1 Hierarchies, Naming, and Types of Policy 70
5.4.2 Motivation for ABAC in Hyperledger Fabric Blockchain Network 70
5.4.3 Coarse Access control Scenario in HLF 71
5.5 System Architecture and Implementation 72
5.6 Use Case And Scenarios 73
5.6.1 Policy Machine Implemented for gREIT 75
5.6.2 gREIT Access Request Scenarios 76
5.7 Performance Evaluation of Policy Machine Implementation in Hyperledger Fabric . 77

Chapter 6: Conclusion & Future Work 80

Bibliography 81

Vita

LIST OF TABLES

2.1 Basic Model Elements & Relations. 17

3.1 A formal expression, no delete operation is required to grant access authorization 27

3.2 Precondition Association & Auth. Enabling User Attributes 30

3.3 Scenario-II Policy Elements & Relations. 34

3.4 Proportion of Nodes for Scenario-I and Scenario-II 42

4.1 Scope of Authorization/Revocation on Attribute Groups 46

4.2 Generic access authorization approaches for *Cathy*'s request 51

4.3 Constrained access authorization approaches for *Cathy*'s request, without set $\mathtt{UA}_3$ 52

4.4 Constrained access authorization approaches for *Cathy*'s request, without set $\mathtt{UA}_2$ 52

LIST OF FIGURES

2.1 Core ABAC Representation (adapted from [68]) 14

2.2 Policy Machine Authorization Graph 15

3.1 A Generic Authorization Graph: *source user* requesting administrative operation on a user attribute 29

3.2 A Generic Authorization Graph: *source user* requesting administrative operation on an object attribute 32

3.3 Policy Machine Authorization Graph 37

3.4 Distribution of Response Time to Generate Access Authorization Approaches for Scenario-II on graphs with 1,000 nodes 42

4.1 Policy Machine Authorization Graph 44

4.2 Average latency for number transactions and access requests 56

5.1 A Ledger L comprises Blockchain B and world state W 64

5.2 Network infrastructure partitioned by member organizations and their connection to a channel 66

5.3 The transaction flow for query and update proposal to the ledger 69

5.4 Partial Components of a Channel, channel01 71

5.5 World state W1 for Ledger L1 contains two business objects 71

5.6 Blockchain Access Control System Architecture. 72

5.7 Policy Machine Authorization Graph for global REIT 74

5.8 Policy Decision Smart contract response to access request scenarios 74

5.9 Average latency of register, launch, and buy transactions using LevelDB . . 77

5.10 Average latency of register, launch, and buy transactions using CouchDB . 78

CHAPTER 1: INTRODUCTION & MOTIVATION

The first step before a user can operate on protected resources in the computer system is that it asserts the user's identity. Usually, the protected resources such as directories, files, applications, data, e.t.c., are called objects. A common name for a user requesting access to objects and system processes acting on the user's behalf are called subjects.

An individual or organization that owns resources of value in the computer system has the authority to establish a policy that describes what operations may be performed upon those objects, by whom, and in what circumstance those subjects may perform those operations. If the subject satisfies the access control policy established by the object owner, then the subject is authorized to perform the desired operation on that object-better known as being granted access to the object. The operations(actions) a permitted user can perform on an object, for example a file directory, includes create, read, update, and delete of file(s). If the subject does not satisfy the policy, then it is denied access to the object.

Access control policies are high-level stipulated conditions that define how access is managed and who, under what circumstances, may access what information. Though access control policies may be application-specific, access policies usually involve subject actions within a specific system environment or throughout an organization's perimeter. For example, policies may deal with the usage of an object within or across the entire organization units or may be based on need-to-know, competence, authority, obligation, or conflict-of-interest factors [27]. These policies can span more than one computing platforms and applications. The enforcement of access policies and governance of information sharing within the organization requires codifying an organization's specified access policies into machine-enforceable algorithms or access control languages.

Access control models are formalized computing algorithm or mathematical representation of the security policies enforced by access control systems. It provides vital insights for proving theoretical limitations of the access control systems. Access control models are the connecting link for a rather wide gap in abstraction between the policy and the mechanism. The use of access control models to represent access control policies serves as a means to incorporate formalization, effectively communicate when describing the characteristics of an access control system, and alleviate the difficulty of access control policy specification.

Since the development of access control models in the early seventies, the three most widely adopted models are the Discretionary Access Control (DAC) [59], Mandatory Access Control (MAC) [59], and Role-Based Access Control (RBAC) [20, 57, 58].

Attribute-Based Access Control (ABAC) [26, 28] is the current milestone in access control models, is a model that regulates access permissions based on the attributes (characteristics) of the subject, resources, and in some cases, the context (or environment). ABAC subsume the three previous traditional models, provides dynamic and effective access control decisions by involving associated attributes of subject and object in making a decision. Also, ABAC offers a powerful and flexible access control by allowing an unrestrained number of user and object attributes in access control decisions.

Attribute-based access control allows organizations to enforce access decisions based on the attribute of the subject, resource, action, and environment involved in an access event.

- **Subject** The subject is the user requesting access to a resource to perform an action. An ABAC system collect data from human resource directory or information from authentication tokens at the time of login to create attributes that characterize the user/subject. These attributes form a profile for the user and are user ID, job roles, user groups, unit and organization memberships, management level, security clearance, and other distinguishable characteristics.

- **Resource** The resource is the protected asset or object such as file, database, application or Application Programming Interface that the subject wants to access. Properties of a resource that serves as attributes includes creation date, owner, name, type, and sensitivity of the resource. For example, when a student tries to access an online grade, the resource involved is “student grade = <valid ID number>”

- **Action** The action is whatever the requesting user do with the resource. Typical action attributes are read, write, edit, copy, create, execute, delete. Some system environments require customized (multiple) attributes to describe an action. Before a student have an online access to grades in the previous example, the action of a lecturer requesting to post the grade can have the characteristics “action type = post”.

- **Environment** The environment is the general situation at the time of every access request. Contextual considerations that represent environmental attributes are time and location of a requested access, the subject’s device, and communication protocol. Organization can include established risk signals such as previous subject’s pattern of behavior and authentication strength as part of the contextual information.

1.1 The Need For Attribute-Based Access Control

Access control policy specification without the use of attributes and context can offer a reasonable level of granularity. It becomes very challenging to maintain these polices in the heterogeneous and dynamic enterprise landscape of this moment and the coming future. There are difficulties in managing traditional access control policies in the following cases:

1. There are various rationale such as logic, performance, and security that may require an organization to have multiple databases. Usually, these databases are of different types such as relational, noSQL, and data warehouses, each with its own flavor of access control policies.

2. We are in an era of Continuous Integration and Continues Delivery(Deployment) CI/CD in application development. As a result, newer set of resources are being added to the database all the time. With static policies, it is very hard to ensure that the right policies are in place at all times.

3. There is a growing trend in the move away from the traditional perimeter based security model toward the application of the security model known as zero trust architecture that treat the enterprise intranet (internetwork) as untrusted to the same degree as the Internet.

The need to adapt a dynamic incorporation of multiple factors such as the degree of trust in the user and device, user and device situational context i.e. location, time-of-day, type of task as well as the current security threat level in the userâs immediate environment. An access control system without attribute-based policies can not enforce a zero trust architecture.

1.2 Policy Reviews and Complexities of Attribute-Based Access Control

The extent to which data is protected and shared among the authenticated users in an organization depends on access policy enforcement. Many organizations are increasingly paying attention to sharing and protecting their resource and information. One of the challenges of the access control systems is that its implementation must be tailored to every use case [27]. For instance, some access control capabilities are packaged as part of an entire service product, and others are offered as an appended feature for managing access configurations within or across architectural abstraction.

When the particular access control mechanism that is included within an application or operating system is not critically examined and evaluated, the consequences are costly. This can undermine the administrative and user productivity as well as the organizationâs ability to perform its primary purpose. As an example, a medium-sized enterprise with significant number of subjects, the number of systems and devices that requires access control configuration are in the hundreds, and the number of objects(resources) that requires protection are in the millions.

Some of the administrative difficulties in the adoption ABAC is as a result of large set of attributes [6]. Firstly, it is laborious to assign or de-assign these attribute values to users or objects. Secondly, authorization policies definition using attributes would be large and complex in nature, making specification and modification difficulty. Imagine a scenario of a misconfiguration of single permission, a user is definitely given an unintended access to information and systems or is ineffective in performing his/her duties. The bottom line is that security posture of the organization has been weakened.

In addition, the interaction of access control functionality with other application behavior can make it difficult to understand, analyze, and evolve access control functionality [52]. Policy reviews are critical to the specification of a new policy based on existing policies or updating an existing policy. For example, to define a new policy that allows ‘manager’ to approve a ‘new loan’, an administrator may first want to check, who (in terms of user attributes and values) can approve ‘new loan’ for existing set of policies. In other words, policy reviews can prevent or mitigate the consequences of policy misconfiguration. System administrator can compare the outcomes of query (policy review) with expected result for a specified policy.

1.3 Problem Statement

Common policy review questions in an ABAC system are the evaluation of the capabilities associated with users and their attributes and the access control entries of objects and object attributes. For logically formulated ABAC, the satisfiability of these typical policy queries is an NP problem but a polynomial-time or linear-time problem in enumerated ABAC policies. The hierarchical structure of an enumerated ABAC model allows specifying a policy in multiple ways. A subset of the various approaches of expressing a policy may complicate or contradict an existing policy. Answering unconventional but important policy review questions can prevent unintended consequences that may arise from multiple approaches for policy specification.

In the context of enumerated ABAC, we consider the following policy review questions that are vital to policy updates:

1. What are the policy elements (user and object attributes) that can enable access of a user to a given resource?

2. What are the policy elements (user and object attributes) that can enable access revocation to a user over a given resource?

3. Can a user attribute confer/delegate granted access rights to other user attributes?

4. If there exists a chain of conferred/delegated access rights, then what is the depth of the chain?

In this work, We will leverage on answering the first two of the previous ABAC policy review questions to develop algorithms that generate approaches to authorize and revoke access in the enumerated ABAC model.

1.4 Summary of Contributions

The summary of what this work contributes is the following:

1. We develop and implement an algorithm that reviews an ABAC policy and apply the reviews to generates all the possible relations that can grant access for a denied administrative access requests.
2. We extend the on previous algorithm by allowing a set of parameters to provide additional flexibility in authorization and revocation of access in an ABAC policy environment.
3. With a use case scenario, we implement the variations of the ABAC policy review algorithms in Hyperledger Fabric, a framework for the blockchain application development.

1.5 Organization of Dissertation

The chapter 1 of this dissertation introduces the concept of access control model in computer systems. In chapter 2 we provide general background to Attribute-Based Access Control and that of an instance, Policy Machine (PM). The algorithms that generate generic approaches to grant access through ABAC policy reviews were presented in chapter 3. A refinement and extension of the algorithm in the preceding chapter was discussed in chapter 4. As a proof of concept, proposed algorithms in this dissertation are implemented in chapter 5. Concluding, chapter 6 provides possible future research that can be done in ABAC policy review.

CHAPTER 2: BACKGROUND AND LITERATURE SURVEY

2.1 Access Control Models

An assigned computer administrator utilize access control mechanisms in logic aligned to protect owners' objects by mediating requests from subjects. These access control mechanism can use a variety of methods to enforce the access control policy that applies to those objects. How these mechanisms function can be described in terms of various logical access control models. These access control models provide a framework and set of boundary conditions upon which the objects, subjects, operations, and rules may be combined to generate and enforce an access control decision. Each of the access control models has its advantages and limitations.

Access control model evolution began with the dichotomy between governmental and commercial needs that led to the development of two distinct access control mechanisms, Mandatory Access Control (MAC) and Discretionary Access Control (DAC). Limitations of MAC and DAC motivates the development of Role-Based Access Control. Role-Based Access Control (RBAC) is considered a much more generalized model than either MAC or DAC, encompassing both models as special cases while providing a policy-neutral framework that allows RBAC to be customized on a per-application basis.

The current milestone in access control models is the Attribute-Based Access Control (ABAC) model. ABAC model authorizes access to perform a set of operations by evaluating attributes associated with the subject, object, and in some cases, environment conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes [68].

2.1.1 Discretionary Access Control (DAC)

Discretionary Access Control (DAC) is an authorization-based approach that policies control access based on the identity of the requestor and on rules stating what requestors are (or are not) allowed to do.

The access matrix model is a framework for describing a primitive discretionary access control policy. In the access matrix model, the state of the system is defined by a triple (S, O, A), where S is the set of subjects, who can exercise privileges; O is the set of objects, on which privileges can be exercised (subjects may be considered as objects, in which case $S \subseteq O$); and A is the access matrix, where rows correspond to subjects, columns correspond to objects, and entry A[S, O] reports the privileges of S on O. The type of the objects and the actions executable on them depend on the system. Changes to the state of a system is carried out through commands that can execute primitive operations on the authorization state, possibly depending on some conditions.

Although the matrix represents a good conceptualization of authorizations, it is not appropriate for implementation. In a general system, the access matrix will be usually enormous in size and sparse (most of its cells are likely to be empty). Storing the matrix as a two-dimensional array is therefore a waste of memory space. Two of the approaches to implementing the access matrix in a practical way are:

- **Access Control List (ACL)**: The matrix is stored by column. Each object is associated with a list indicating, for each subject, the actions that the subject can exercise on the object.
- **Capability**: The matrix is stored by row. Each user has associated a list, called capability list, indicating, for each object, the accesses that the user is allowed to exercise on the object.

2.1.2 Mandatory Access Control (MAC)

Mandatory (MAC) policies control access based on mandated regulations determined by a central authority.

DAC are ideally appropriate for systems and applications in the commercial and industrial settings because of its flexible policies. However the DAC policies have a costly disadvantage by not providing assurance of information flow in a system. Specified authorization is easily circumvented. For instance, a user granted a read access on a data can pass this authority to another user without the knowledge of owner - information dissemination is not controlled. Conversely, MAC systems controls information dissemination by preventing the flow of information from high-level objects to low-level objects. Access in MAC is govern on the basis of system's subjects and objects classification. There is an access class assignment for every subject and object in a mandatory system. The set of access classes is a partially ordered set and usually a security level and a set of categories constitute an access class. Logic and use of the access classes assigned to objects and subjects in an application of a multilevel mandatory policy determines if the access class is meant for a secrecy or an integrity policy. That is, MAC is implemented as a secrecy-based and integrity-based mandatory policies.

2.1.3 Role-Based Access Control (RBAC)

Role-based (RBAC) policies control access depending on the roles that users have within the system and on rules stating what accesses are allowed to users in given roles.

In the early 2000s, the National Institute of Standards and Technology (NIST) proposed a U.S. national standard for role-based access control through the International Committee for Information Technology Standards (ANSI/INCITS) [20] introduced a family of models called RBAC$_1$ to RBAC$_3$.

The RBAC$_0$ (core RBAC) is the base model that consist of users, roles, objects, operation, and permissions as basic elements. Users access are indirectly granted through an assignment of users to roles, and permissions (i.e., the association between an object and an operation excitable on the object) to roles. RBAC$_0$ offers flexibility through the many-to-many relational assignments of permission-to-role and user-to-role. Also in RBAC$_0$, is the concept of session, where each session is a mapping between a user and a (subset of roles assigned to the user) set of activated roles. Thus, available permissions to a user for a given session are all the permissions associated with the roles activated by the user in the session.

The other RBAC models introduce role hierarchies (RBAC$_1$) and constraints (RBAC$_2$) [55]. A role hierarchy is a partial order on roles that lets an administrator define that one role is senior to another role, which means that the more senior role inherits the junior role's permissions. For example, if a Manager role is defined to be senior to an Engineer role, any user assigned to the Manager role would also have the permissions assigned to the Engineer role.

Constraints are predicates over configurations of a role model that determine if the configuration is acceptable. Typically, role models permit the definition of mutual exclusion constraints to prevent the assignment of the same user to two conflicting roles, which can enforce separation of duty. Other constraints that are frequently mentioned include cardinality constraints to limit the maximum number of users in a role, or prerequisite role constraints, which express that, e.g., only someone already assigned to the role of an Engineer can be assigned to the Test-Engineer role. The most expressive model in the family is RBAC_3, which combines constraints with role hierarchies.

2.1.4 Attribute-Based Access Control (ABAC)

Attribute-Based Access Control (ABAC) An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environmental conditions, and a set of policies that are specified in terms of those attributes and conditions.

Attribute-Based Access Control (ABAC) [28, 68] has been around for over two decades. Numerous ABAC models have been proposed [2, 30, 47, 61, 63, 71]. Despite the existence of these different ABAC models, there is no consensus on a specific standard ABAC model. However, a well-accepted simplest form of attribute-based access control includes users, user attributes, objects, object attributes, actions, and permissions or operations allowed for users on specific objects, based on attributes of users and objects. This typical foundation provides great freedom to researchers to incrementally enhance this basic ABAC architecture based on customized needs and requirements in different scenarios. Any additional component that uses or is compatible with the basic ABAC components can be incorporated with them in order to get a more powerful and flexible ABAC model.

As limitations of the two foremost traditional access models (DAC and MAC) inspires the design of RBAC, role-based access control has its own set of limitations such as role explosion and role-permission explosion [49]. It is also restrictive in nature since accesses are based only on roles and it is difficult to include other characteristics of users, and contextual or environmental factors (e.g. time, location, etc.) in access control policies.

On the other hand, attribute-based access control policies can easily incorporate these factors with environmental and contextual attributes [2,61], as well as define access control policies based on different characteristics of users and objects, besides roles. Although ABAC research has received significant attention in academia, it is not so common to find implementations of these models in the industry.

There are two major techniques for specifying authorization policies in Attribute Based Access Control (ABAC) models [43]. The more conventional approach is to define policies by using logical formulas involving attribute values. Examples in this category include ABACα, HGABAC and XACML. The alternate technique for expressing policies is by enumeration. Policy Machine (PM) and 2-sorted-RBAC fall into the later category.

2.2 Policy Review in Access Control Models

Policy review (policy review query or policy content query) belongs to a class of analysis queries that directly examine policy content [5]. We examine typical policy review in the models discussed in the preceding section.

2.2.1 Discretionary Access Control

Early approaches to specifying DAC policy allowed associating conditions and user groups with authorizations to restrict their validity. Conditions are (system-dependent (e.g., time or location), content-dependent (e.g., relation of an object to another), and history-dependent (i.e., previously grated access)) system predicates that if only satisfied is an associated authorization valid. The concept of user groups (e.g., Contractors, Analysts, Auditors) allows the use of group hierarchy in policy specification. Implementations of DAC that support features such as conditions and user groups motivates the following policy review questions: (i) What group of users are authorized to perform a given operation on an object? (ii) What are the chain of objects linked to a given object in a hierarchy?

In ACL, policy review of authorizations associated with an object is a trivial search, while policy query of authorizations associated with a subject or user group requires a relatively complex search of individual ACLs for all objects [15]. Analogously, with capabilities, it is immediate to retrieve queries on the privileges of a subject, while evaluating all the accesses executable on an object requires the search of all the different capabilities [55].

2.2.2 Role-Based Access Control

Administrative reviews is a category in the features that the RBAC system and administrative functional specification defines. The administrative review features define requirements in terms of an administrative interface and an associated set of semantics that provide the capability to perform policy query on RBAC elements and relations. Some of the queries answered by the administrative functions are the following that return the set of:

- users assigned to a given role
- roles assigned to a given user
- operations a given user may perform on a given object
- all permissions either directly granted to or inherited by a given role
- roles directly assigned to a given user as well as those "roles that were inherited by the directly assigned roles."

2.2.3 Attribute-Based Access Control

To demonstrate expressive power and flexibility, several ABAC models including [2, 19, 61, 71] have been proposed in past few years. These models adopt the conventional approach of designing attribute based rules/policies as logical formulas. Using logical formulas to grant or deny access is convenient because of the following reasons. (i) Simple and easy: Creating a new rule for granting access is simple. It does not involve upfront cost like engineering roles in case of RBAC. (ii) Flexible: Rules are easy to succinctly specify even complex policies. There is no limit on how many attributes can be used in a rule or how complex the language be to specify the rule. Given a required set of attributes, and a computational language, ABAC policy is only limited to what the language can express [68].

Interestingly, designing a rich computational language to define attribute-based rules makes policy update or policy review an NP-complete or even undecidable problem. For example, authorization policies in many existing ABAC models including [2,61,71] are expressed in propositional logic. Reviewing policy in these models (which may simply ask, for a given policy which (attribute, value) pairs evaluate the policy to be true) is similar to the satisfiability problem in propositional logic which is NP-complete. Likewise review for policies specified in first-order logic is undecidable.

Another method for specifying attribute-based policies is by enumeration. Policy Machine [12] and 2-sorted-RBAC [37] fall into this category. Enumerated policies can also be very expressive. Ferraiolo et al. [12] show configuration of LBAC, DAC and RBAC in Policy Machine using enumerated policies [12]. Moreover, updating or reviewing an enumerated policy is inherently simple (polynomial time) because of its simple structure. It should be noted that the size of an enumerated policy may be exponential relative to a succinct formula which expresses the same policy. Thus there is a trade-off between these two methods for specifying policies.

2.3 Attribute Based Access Control

Attribute-Based Access Control: An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environmental conditions, and a set of policies that are specified in terms of those attributes and conditions. The ABAC model not only has the flexibility to enforce a combination of the previous models but also has the expressive power lacking in the traditional models.

At the minimum, any ABAC system can evaluate attributes and enforce rules or relationships between those attributes. The Attribute-Based Access Control Mechanism evaluates attributes and

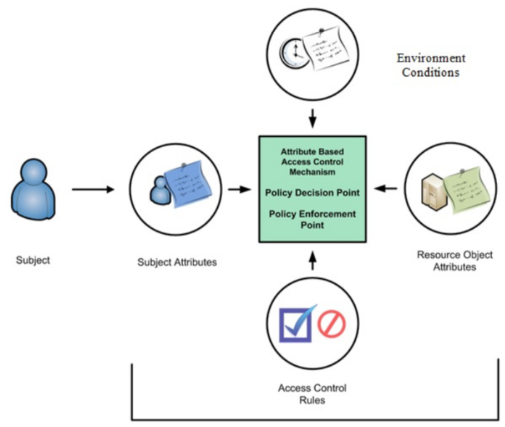


Figure 2.1: Core ABAC Representation (adapted from [68])

access control rules to provide access decisions for all access requests. Essentially, the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP) are components at the core of an Attribute-Based Access Control mechanism. While the Policy Enforcement Point interfaces the access request and protected resources, the Policy Decision Point determines whether to grant or deny access requests made through the PEP based on access control rules (see figure 2.1).

All subjects and objects within the system must be assigned specific attributes that characterize them. In some cases, a single attribute is sufficient to distinguish an entity. For example, identifying an object by ownership ascribed to it. Other scenarios require a combination of attributes to capture a subject or object. For instance, determining a subject based on the organization they belong to, the certifications held, and years of experience. Whenever there is a change to system entities, associated attributes may require updates.

2.4 Attribute-Based Access Control Models

At this moment, a metamodel instance CABAC [19] is one of the most recent efforts to formalize the ABAC model. While there is no standard ABAC model, research effort classified proposed models as a pure or hybrid [62]. The formalism of various ABAC models differs slightly, but they constitute common entities and relations. Some of the proposed pure ABAC models are system agnostic and subsume other traditional models. Another group of proposed pure ABAC models is

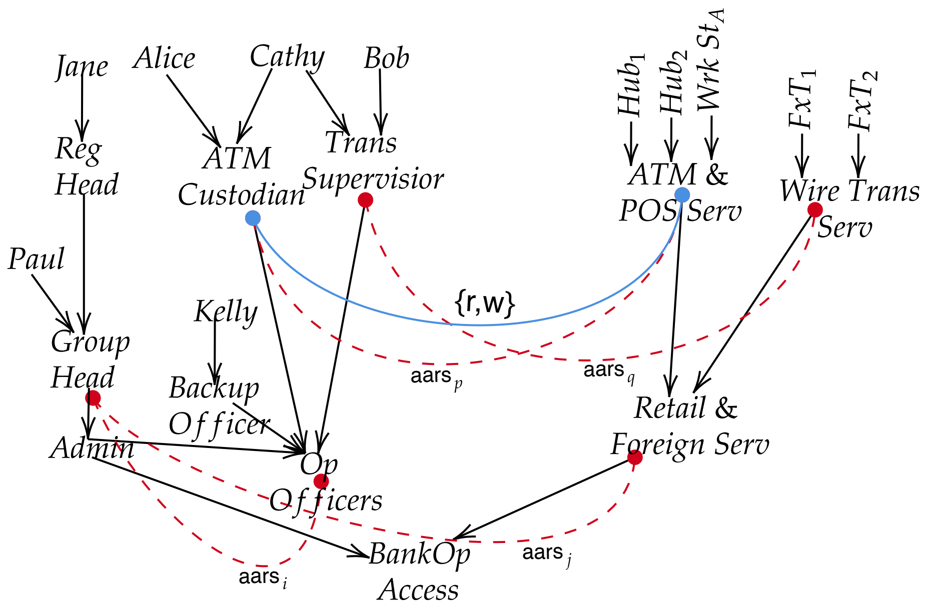


Figure 2.2: Policy Machine Authorization Graph

domain-specific for cloud-computing, mobile environment, grid computing, and web services.

Hybrid ABAC models aim to integrate attributes into existing models of access control to extend traditional models with an identity-less or policy-based access control notion. An early hybrid approach towards an ABAC model was the introduction of parameterized roles and permissions to RBAC, PRBAC [69]. Other RBAC-ABAC hybridizations are Attribute-Based Role Assignment [29], Attribute-Centric [18, 36], and Role-Centric [33]. Apart from the RBAC extension, a recent effort was to unify ABAC with alternative access control models [32].

An ongoing effort to standardize Attribute-Based Access Control by the (American National Standards Institute / International Committee for Information Technology Standards) ANSI/INCITS is the Next Generation Access Control (NGAC) standardization. The Policy Machine (PM) is the basis for the Next Generation Access Control. We implemented a couple of the Attribute-Based Access Control policy review issues previously discussed in the context of the policy machine.

2.5 Policy Machine Overview

The rest of this section provides an overview of policy machine core data elements and selected relations pertinent to this work. The assignment, association, prohibition, and obligation are the primary relations. Privileges and restrictions are derived relations from associations and prohibitions in the PM respectively. In this work, we only focus on the assignment and association relations and concepts related to them. To ease referencing, we provide the formal definition of PM components discussed in this work in table 2.1. Readers with prior knowledge of policy machine fundamentals may skip this section.

PM Basic Elements And Relations

Policy specification in Policy Machine has an annotated Directed Acyclic Graph (DAG) representation. Basic elements of the PM called Policy Elements (PE) are the nodes in the access control graph. These nodes are the finite sets of Users (U), Objects (O), User Attributes (UA), Object Attributes (OA), and Policy Classes (PC). As shown in the access control graph of figure 3.3, the users / user attributes are the left top nodes. Unlabeled directed acyclic edges called assignments are relations allowed from user nodes to user attribute nodes, user attribute to user attribute nodes, and user attribute to policy class node. In a similar fashion on the right side of the graph, connected unlabeled directed acyclic edges start from object / object attribute nodes, through object attribute nodes, and terminates at the policy class node. All the nodes except for the policy class must have a path to the policy class. Users' access to protected resources is only possible through the creation of an association. An association is a relation represented by labeled (annotated) downward-arcing edge from a user attribute node to an attribute (user attribute or object attribute node). For instance, in figure 3.3, the association triple (*Group Head*, $aars_i$, *Retail & Foreign Serv*) specify that a user who has a path to *Group Head* is authorized to perform operations enabled by $aars_i$ on *Retail & Foreign Serv* and policy element that has a path to *Retail & Foreign Serv*. Access granted through an association could be a set of resource access rights or a set of administrative access rights (shown in solid blue and dashed red edges respectively). The policy elements and the relations constitute

Table 2.1: Basic Model Elements & Relations.

U	A finite set of authorized users; U = $\{u_1, ..., u_n\}$ where u_i denotes a member of U.
P	A finite set of system processes; P = $\{p_1, ..., p_n\}$ where p_i denotes a member of P.
O	A finite set of protected objects; O = $\{o_1, ..., o_n\}$ where o_i denotes a member of O.
UA	A finite set of user attributes; UA = $\{ua_1, ..., ua_n\}$ where ua_i denotes a member of UA.
OA	A finite set of object attributes; OA = $\{oa_1, ..., oa_n\}$ where oa_i denotes a member of OA. OA $\supseteq$ O
PC	A finite set of policy classes; PC = $\{pc_1, ..., pc_n\}$, where pc_i denotes a member of PC.
PE	A finite sets of users in U, objects in O, user attributes in UA, object attributes in OA, policy classes in PC; PE = U $\cup$ UA $\cup$ OA $\cup$ PC and O $\subseteq$ OA
AT	The finite set of user and object attributes; AT $\overset{\text{def}}{=}$ UA $\cup$ OA.
AR	A finite set of resource access rights RAR and administrative access right AAR; AR = $\{ar_1, ..., ar_n\}$ and 2_1^{AR} is the non empty subsets of AR, where ar_i denotes a member of AR. AR = RAR $\cup$ AAR
ASSIGN	The assignment relation is a binary relation on the set of policy elements, PE. ASSIGN $\subseteq$ (U$\times$UA) $\cup$ (UA$\times$UA) $\cup$ (OA$\times$OA) $\cup$ (UA$\times$PC) $\cup$ (OA$\times$PC)
ASSIGN$^+$	The binary relation ASSIGN$^+$ is the transitive closure of the assignment relation ASSIGN. ASSIGN $\subseteq$ ASSIGN$^+$
ASSOCIATION	The ternary relation ASSOCIATION from UA to 2_1^{AR} to AT. ASSOCIATION $\subseteq$ UA $\times$ 2_1^{AR} $\times$ AT If segregated by RAR and AAR, ASSOCIATION $\subseteq$ (UA $\times$ 2_1^{AR} $\times$ AT) $\cup$ (UA $\times$ 2_1^{AR} $\times$ AT)

the authorization graph. The table shown in the figure is a formal definition of the PM components mentioned and others in subsequent sections of this dissertation.

2.6 ABAC Policy Specification Methods

2.6.1 Enumerated Policy

Two related work to policy review in the PM framework attempt to answer two types of user queries. One, can a particular user operate on an object? Two, what privileges does a user have? As an example, "can Carl review all objects in the system that he can read or write?"

Mell et al. [45] was the first attempt to implement graph search approaches to query policy in the PM .The other contribution made use of a variation of an algorithm that finds common predecessors of two nodes in a Directed Acyclic Graph (DAG). A comparison of the performance characteristics of the techniques in the mentioned work to the existing implementations shows significant improvement. The time complexity to evaluate access decisions and review access rights based on the number of edges improved from quadratic to linear. In the case of Basnet et al. [4], it is linear in the number of operation nodes (they converted operation labeled edges into nodes) for making access decisions and log-linear in the number of user and operation nodes to review access rights.

In contrast, we apply policy queries to create or modify rules. That is, we evaluate policy queries to create rules that grant or deny user access.

2.6.2 Logical-Formula Policy

The PTaCL and XACML are some languages that support logic-based formula for expressing ABAC policy, where attribute names/values pairs are terms for defining policies and access requests. Policy languages expressed in propositional logic apply an ABAC policy to a request by matching the attributes in the request with policy attributes. At the time a request is evaluated, some attribute values are retrieved incorrectly. The consequence is a complex decision that describes all possible evaluation outcomes to make up for missing attributes. A novel evaluation mechanism that uses a non-deterministic attribute retrieval for a given request addresses the complex decision problem for the enforcement mechanism that needs to resolve it into a conclusive one [11].

Zhang et al. present a model-checking algorithm to evaluate access control policies and a tool as its implementation [72]. The algorithm checks whether the access policies allow authorized users to reach their goals and prevent intruders from achieving their malicious actions. The authors utilize their previous work that formalizes the RW access control system description and verification framework to describe the policies and goals (read and overwrite operations) of agents (subjects) evaluated. The algorithm performs two modes of checking on its input, a policy description, and a goal. The accessing mode of the algorithm searches for strategies consisting of reading and writing steps that allow the agents to achieve their operations regardless of the system state. In the intrusion detection mode, the algorithm evaluates the willingness of intruders to guess the value of attributes that s/he not allowed to read.

2.7 Administrative Role Based Access Control Policy Analysis

2.7.1 ARBAC97

Administrative role-based access control ‘97 (ARBAC ‘97) [56] defines administrative roles and specifies how members of each administrative role can change the RBAC policy. It enables the specification of policy for user-role assignment changes. Also, ARBAC requires separate administration [66]. In other words, the administrative role cannot serve as the target role in the rules that expressed the policy. However, the work of [70] argues that the assumption of separate administration limits the expressiveness and applicability of ARBAC.

There are several research contributions on the user-role reachability analysis of ARBAC. These works query whether a given user can be assigned to given roles by given administrators. Studies on the analysis of user-role reachability [22, 23, 31] assume separate administration by following the ARBAC ‘97 definition. Stoller et al. analyzed the ARBAC user-role reachability without separate administration using an algorithm that allows roles that appear negative and positive in the policy and a fixed tractable number of users. However, the algorithm may have exponential complexity for the users and mixed roles but have polynomial complexity for a fixed input size in numbers of users and mixed roles. By reducing the number of ARBAC rules and users considered during analysis, some research efforts [70] optimized Stoller and Co algorithm.

2.7.2 Administrative TRBAC

There are two temporal extensions of the RBAC model, TRBAC, which impose temporal constraints on roles activation or user-role assignment. Also, these models restrict the ability to perform administrative action by temporal constraints. The administration of extended RBAC policies with contextual information, like time and space, is complex since they must fulfill fluid requirements. There are a couple of research contributions on the safety problem of Administrative Temporal RBAC (ATRBAC). Analysis in these studies identified administrative actions that generate policies through which a user can acquire permissions that may compromise some security goals. Mondal et al. [41] reduced the safety problem for ATRBAC policies to a verification problem of timed automata [3]. Apart from safety, their approach supports other verification properties but assumes a fixed set of users. The disadvantage is that whenever the set of users changes, the analysis requires a re-run. Furthermore, it is not a scalable technique because the state space exploration grows exponentially by the model checker in the number of users.

Another approach proposed in [67] transformed the safety problems of ATRBAC policies into reachability problems for policies expressed in models that are similar to URA97 or URA97 variant. Their work allows leveraging previous effort in analyzing Administrative RBAC policies by using existing tools such as RBAC-PAT [22] or VAC [21]. The drawback of [67] is the state-space explosion, there is a need to solve several safety problems for URA97, and the complexity of many constrained versions of this problem is NP-hard [60]. Ranise et al. [51] apply the symbolic model checking technique (that uses a class of the first-order formulae, Bernays-Shonfinkel-Ramsey BSR [50]) to solve safety problems for ATRBAC policies and addressing the limitations of previous contributions mentioned. Their work translates the safety problem of ATRBAC policies to a reachability problem of BSR transition systems.

2.8 Attribute Based Access Control in Hyperledger Fabric

Here, we overview previous research contributions on exploiting blockchain for an ABAC implementation. The application of blockchain as an Attribute-Based Access Control system is studied in different domains. The implementation of blockchain-based ABAC in IoT systems is discussed by Pinno et al. [46], Ding et al. [16], and Dukkipati et al. [17]. Zhang and Posland in [73] propose a blockchain-oriented authorization scheme for Electronic Medical Records (EMRs). The granularity of their authorization approach support queries of blocks and attribute values. Also, their method lowers the computational overhead by eliminating the use of the Public Key Infrastructure (PKI) for authorization, encryption, and decryption.

Another attempt to control access to Electronic Health Record (EHR) proposes an architecture that uses both blockchain and edge node [24]. The blockchain serves as the controller that manages the identity and access control policies specified in the Abbreviated Language For Authorization (ALFA). Besides, the edge node is off-chain storage for the EHR data and uses specified policies to enforce attribute-based access control on EHR data in combination with blockchain-based access control logs.

Maesa et al. [39] introduced an access control service utilizing the Ethereum blockchain platform. The blockchain stores smart contracts as access control policies specified using eXtensible Access Control Markup Language (XACML) [40] and process access decision making. Gou et al. [25] propose a multi-authority attribute-based access control scheme by using Ethereum's smart contracts. The Ethereum smart contracts provide the specification for the interactions between data owner, data user, and multiple attribute authorities. A data user presents its attributes to different attribute authorities and receives attributes tokens from respective attribute authorities only if validation of attributes is successful. The use of a distributed Attribute-Based Access Control system based on Hyperledger Fabric blockchain to provide trusted auditing of access attempts was proposed by Rouhani et al. [53].

There are few implementations of attribute-based access control on a blockchain for a domain-independent scenario [25, 39, 53]. While [39, 53] specify policies using the XACML, we utilize the Policy Machine, a different open-source attribute-based access control framework developed by NIST. Rouhani et al. [53], the only generic implementation of attribute-based access control on Hyperledger Fabric blockchain network, utilizes ABAC components as smart contracts to control access to an off-chain system. In contrast, our attribute-based access control implementation specifies policies using smart contracts for access control to the on-chain data. Also, an all-purpose blockchain-oriented attribute-based access control implemented by Maesa et al. [39] and Guo et al. [25] utilizes the Ethereum blockchain platform. The Ethereum based blockchain implementation is costly because of the fee paid for every operation, but there are no fees in the permissioned blockchain, such as the Hyperledger Fabric.

CHAPTER 3: ADMINISTRATIVE ACCESS AUTHORIZATION THROUGH POLICY REVIEW

An access request in the policy machine model is a request to operate on resources represented as objects (i.e., non-administrative access request) or on the creation and maintenance of policy elements and relations (i.e., administrative access request). For both access request types, we consider authorizing access requests through subsets of administrative privileges. In other words, a user who is not a superuser (root user) can grant access to another user. The relationship between a user requesting access and the resource referenced by that request dictates the authorization approaches. In this chapter, we discuss the access request scenarios, the policy review of policy elements involved in granting access, and how we apply these scenarios and the reviewed policy elements in our algorithm.

3.1 Preliminaries

3.1.1 Definitions

The policy review of authorization approaches can be stated as – given a denied access request, find all the possible sets of one or more access request sets that can authorize the denied access request such that no strict subsets of any access request set can grant the denied access.

Formally, given $\text{AREQ}_D : \forall \texttt{areq}_d \in \text{AREQ}_D \cdot \big(\text{Access\_Decision}(\texttt{areq}_d) = \textit{deny}\big)$, find AREQ_{auth} :

$\forall \texttt{areq}_d \in \text{AREQ}_D, \forall areq_A \in \text{AREQ}_{auth} \cdot \Big(\big(\forall areq \in areq_A \cdot \text{Access\_Decision}(areq) = \{\textit{accept}\}\big) \longrightarrow \text{Access\_Decision}(\texttt{areq}_d) = \{\textit{accept}\}\Big)$
$\wedge \Bigg(\forall areq_s \subset areq_A \cdot \Bigg(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\textit{accept}\}\big) \longrightarrow \text{Access\_Decision}(\texttt{areq}_d) = \{\textit{deny}\}\Bigg)\Bigg)$

An Authorizing Access Request is a function that takes a set of denied access requests as input and return a set of one or more access request sets as output for each denied access request such that the following conditions are satisfied:

1. A given denied access request is authorized if all access requests of the associated access request set is authorized.

2. A given denied access request cannot be authorized by any strict subsets of the associated access request set.

A formal expression for this function and its conditions are given as **func_aareq** : AREQ$_D \longrightarrow 2^{2^{AREQ}}$,

Where $\forall \texttt{areq}_d \in \text{AREQ}_D$, $\forall areq_A \in \text{func\_aareq}(\texttt{areq}_d) \cdot \Big(\big(\forall areq \in areq_A \cdot \text{Access\_Decision}(areq) = \{\textit{accept}\}\big) \longrightarrow \text{Access\_Decision}(\texttt{areq}_d) = \{\textit{accept}\}\Big)$

$\wedge \Bigg(\forall areq_s \subset areq_A \cdot \Big(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\textit{accept}\}\big) \longrightarrow \text{Access\_Decision}(\texttt{areq}_d) = \{\textit{deny}\}\Big)\Bigg)$

3.1.2 Assumption

All users in the policy machine derive privileges to operate on the protected resource through assignment and association relations. The assignment relations among attributes of the same type are hierarchical. Thus, the policy machine allows specifying authorization in multiple ways. In other words, creating a combination of relations can grant a single access request. However, there may be a restriction on obvious approaches to permit access. Our algorithm generates an exhaustive list of all possible ways to authorize a request, especially approaches that are manually difficult to identify.

3.1.3 Observations

1. The Principal Authority (PA), also known as the superuser, is a compulsory predefined entity of the PM. The PA is responsible for creating and controlling the policies of the PM in their entirety and inherently holds universal authorization to carry out those activities within the PM. The access rights held by the principal administrator can be delegated to a domain or subordinate administrators except the following:

   (a) The access right to create and delete policy class.

   (b) The access right to create and delete assignment of attributes to the policy class.

   For instance, the Principle Administrator (PA) that instantiated the authorization graph of figure 3.3 is not visible in the graph. He created the policy class (*BankOp Access*), the nodes (*Admin*, *Op Officers*, *Retail & Foreign Serv*, *Group Head*, *Regional Head*, *Jane*, and *Paul*), and the edges between them. At the minimum, he also created the associations ((*Group Head*, $aars_i$, *Op Officers*) and (*Group Head*, $aars_j$, *Retail & Foreign Serv*)). Access rights granted to users with a path to *Group Head* are sufficient to create all other policy elements and relations.

2. To preserve the properties of the policy machine, it requires deleting relations before the policy element. For example, if a user attribute is involved in an assignment or association relation, the user attribute cannot be deleted until it is no longer involved in the relation.

3.1.4 Notations

In this paper, the notation to represent administrative access rights is no different from the generic specification of NIST. We adopt a subscript notation to describe the relationship of a user and process. And dot notation is applied in referencing subentities of an object.

- **User's Process:** A user can have a one to many relations with processes, but a process can have a one to one relation with the user. We denote a user u_s associated with a process p as p_{u_s}.

- **Access Rights:** Every administrative access right for the creation and deletion of policy element is prefixed by 'c-' and 'd-' respectively. A policy element or a pair of policy elements follows. For instance, 'c-$uapc$' is the administrative access right to create an assignment from a user attribute (ua) to a policy class (pc).

- **Access Request Element:** Let $\texttt{areq}_d$ = (p_{u_s}, $\texttt{ar}_x$, $\texttt{target}$) be a tuple of an administrative access request. For simplification, we adopt the dot notation to reference the access request variables. For example, the administrative operation $\texttt{ar}_x$ in the access request denoted as $\mathbf{areq}_d\mathbf{.ar}_x$

3.1.5 Claim

The create and delete access rights are the only types of administrative access rights that applies to policy elements and their relations. However, deletion of any policy element and/or relations can not authorize any denied access request

The deletion of any policy element and/or relations can not authorize any denied access request.

That is to say, given an access request set that authorizes a denied access, if there exist an operation requesting the deletion of any policy element in the access request set and no strict subset of the access request set can grant the denied access, then the denied access is authorized without the operation requesting the deletion of a policy element and no strict subset of the access request set can grant the denied access. We can express this claim formally as in table 3.1.

3.1.6 Scope

The PM framework has multiple facets – the policy elements and the assignments that make up a policy element diagram, the association and prohibition that apply the policy element diagram to form the authorization graph, and obligations that are carried out when access-related events occur [12]. Prohibition and obligation are outside the scope of this work.

Table 3.1: A formal expression, no delete operation is required to grant access authorization

$$\forall \texttt{areq}_d \in \text{AREQ}_D, \forall areq_A \in \text{func\_aareq}(\texttt{areq}_d) \cdot \Bigg(\bigg(\Big(\forall areq \in areq_A : \exists \textbf{d-*} \in \textbf{areq}.\textbf{aop} \cdot$$
$$\text{Access\_Decision}(areq) = \{accept\} \Big) \wedge \text{Access\_Decision}(\texttt{areq}_d) = \{accept\} \bigg) \wedge$$
$$\bigg(\forall \textbf{areq}_\textbf{s} \subset areq_A \cdot \Big(\big(\forall areq \in \textbf{areq}_\textbf{s} \cdot \text{Access\_Decision}(areq)$$
$$= \{accept\} \big) \wedge \text{Access\_Decision}(\texttt{areq}_d) = \{deny\} \Big) \bigg) \longrightarrow$$
$$\bigg(\Big(\forall areq \backslash areq_{dd}\text{: } areq_{dd}.\textbf{aop} = \textbf{d-*} \in areq_A \cdot \text{Access\_Decision}(areq)$$
$$= \{accept\} \Big) \wedge \text{Access\_Decision}(\texttt{areq}_d) = \{accept\} \bigg) \wedge$$
$$\bigg(\forall \textbf{areq}_\textbf{s} \subset areq_A \cdot \Big(\big(\forall areq \in \textbf{areq}_\textbf{s} \cdot \text{Access\_Decision}(areq)$$
$$= \{accept\} \big) \wedge \text{Access\_Decision}(\texttt{areq}_d) = \{deny\} \Big) \bigg) \Bigg)$$

3.2 Policy Review For Access Authorization

We adapt the following terms in the construct of the policy reviews for access authorization:

- A *privileged user* is a user that can grant the permission another user is seeking. Our algorithm only considers granting access by a nonprinciple administrator through the creation of edges (relations) between existing nodes (policy elements).
- A user requesting access to operate on a policy element is the *source user*.
- A protected resource that a *source user* is requesting to operate on is a *target policy element*.
- The *precondition associations* are association relations that a *privileged user* derives its permission to authorize a *source user*.
- A set of policy elements that the creation of one or more (edges) relations between them authorize the *source user*'s request is an *authorization enabling policy element*.

For this work, authorizing access through policy reviews only holds for a *privileged user* granting access by creating edges (relations) between existing nodes (policy elements).

Access Authorization Scenarios

The policy machine access requests are distinguishable by the type of operation or protected resource a subject wants to access. We categorize all access request into two scenarios using the containment relation. This enables us to specify the properties of an *authorization enabling policy element* and the *precondition associations* in our queries.

One possible scenario is when the *target policy element* contains the *source user* (i.e., the *target policy element* is reachable from the *source user*) or there is a user attribute that contains both the *source user* and the *target policy element* (i.e., there is a path from both the *source user* and the *target policy element* to a user attribute). While this first situation only happens if the *target policy element* is a user or a user attribute, the *source user* can only request an administrative operation on these types of *target policy elements*. That is, the *source user* may want to create/delete a user or user attribute node in the authorization graph. Consequently, a user or user attribute *target policy element* limit *authorization enabling policy element* queries to user attribute only.

The second scenario is of a request that the *target policy element* is not reachable from the *source user* and no attribute contains both the *target policy element* and the *source user*. In this case, the *source user* may want to operate on a *target policy element* that is a user, user attribute, an object, or an object attribute. Thus, the access request is an administrative or a non-administrative (resource) access request. For instance, the *source user* may want to read a file, run an application, or create/delete a user, a user attribute, an object, or an object attribute. The operation differentiates the access request in this scenario. For a *target policy element* that is an object or object attribute, the queries for *authorization enabling policy element* includes both (user and object) attributes.

Before we delve into the definition of *authorization enabling policy element* and *precondition associations* for the two scenarios, we adapt following notations for specifying a set and functions in the *authorization enabling policy element*

- `tail` : ASSOCIATION $\longrightarrow$ `UA`: is a function that maps an edge, association relation, $(ua_i, ars_j, au_k) \in$ ASSOCIATION to the (user attribute) node $ua_i \in$ `UA` it originates.

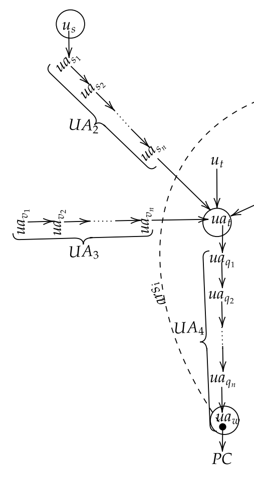


Figure 3.1: A Generic Authorization Graph: *source user* requesting administrative operation on a user attribute

- `head` : ASSOCIATION $\longrightarrow$ `AT`: is a function that maps an edge, association relation, $(ua_i, ars_j, au_k) \in$ ASSOCIATION to the (user/object attribute) node $at_k \in$ `AT` it terminates. Where `AT` = `UA` ∪ `OA`
- `anc`: `PE` $\longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is an ancestor to the policy element.
- `des`: `PE` $\longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is a descendant to the policy element.
- $\texttt{PE}_{i_{\texttt{func}}} = \{\texttt{node} \mid (\exists pe_j \in \texttt{PE}_i)[\texttt{node} \in \texttt{func}(pe_j)]\}$: is the set of all policy elements returned by `func` for the set $\texttt{PE}_i$, where `func` is `anc` or `des`.

Administrative Authorization Enabling User Attributes

For a *source user* whose *target policy element* is a user or user attribute and there is a user attribute that contains the *source user* and *target policy element*, the *precondition association* must satisfy the following:

1. The user attribute node returned by the `tail` function for the *precondition association* as its input is reachable from a *privileged user*.

2. The `head` function returns a user attribute node that is reachable from the *source user*, *target policy element*, and the *privileged user*for the *precondition association* as its input.

Figure 3.1 depicts a generic authorization graph for the first scenario. We assume the *source user* (i.e., u_s) wants to perform an administrative action on a user node, $\mathtt{u}_t$. Since there is no path from the *source user* node to an association, no request is authorized. Also, we assume the administrative access rights set $\mathtt{ars}_i$ activates the operation the *source user* u_s wants to perform on the *target policy element* $\mathtt{u}_t$. Firstly, we query for the *precondition association*, $(\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w)$ in this case. The *precondition association* $(\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w)$ allows a *privileged user* (e.g.,$\mathtt{u}_p$) to create (assignment and/or association) relations that authorizes the *source user*. That is, the *privileged user* creates relation(s) that provides an assignment relation path from the *source user* to the user attribute, $\mathtt{ua}_p$ returned by the `tail` function for the *precondition association*, $(\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w)$. The *privileged user* creates relations using the combination of the following defined user attribute sets, *authorization enabling policy element* to authorize a *source user*.

- UA_1: is a set of user attribute nodes that has a path to the *precondition association* $(\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w)$ predecessor-node ($\mathtt{ua}_p$). That is, the set $UA_1 = \{\mathtt{ua}_{k1}, \mathtt{ua}_{k2}, ..., \mathtt{ua}_{kn}, \mathtt{ua}_p\}$

Table 3.2: Precondition Association & Auth. Enabling User Attributes

$Precondition\ Association$	$\{(\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w) \in \text{ASSOCIATION} \mid ar_x \subseteq \mathtt{ars}_j\}$
$Priv.\,Users$	$\{\mathtt{u}_p \in \text{U} \mid (\mathtt{u}_p, \mathtt{ua}_p) \in \text{ASSIGN}^+\}$
$Source\ Users$	$\{u_s \in \text{U} \mid (u_s, \mathtt{ua}_w) \in \text{ASSIGN}^+ \wedge (u_s, \mathtt{ua}_p) \notin \text{ASSIGN}^+\}$
$Target\ UA$	$\{\mathtt{target} \in \text{UA} \mid (u_s, \mathtt{target}), (\mathtt{target}, \mathtt{ua}_w) \in \text{ASSIGN}^+\}$
UA_1	$\{\mathtt{ua} \mid \mathtt{ua} = \mathtt{tail}((\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w)) \vee \mathtt{ua} \in \mathtt{anc}(\mathtt{tail}((\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w))\}$
UA_2	$\{\mathtt{ua} \mid \mathtt{ua} \in \mathtt{anc}(\mathtt{head}((\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w))) \wedge \mathtt{ua} \in \mathtt{des}(u_s)\}$
UA_3	$\{\mathtt{ua} \mid \mathtt{ua} \in \mathtt{anc}(\mathtt{head}((\mathtt{ua}_p,\ \mathtt{ars}_i,\ \mathtt{ua}_w))), \mathtt{ua} \notin \text{UA}_{2_{\mathtt{anc}}}, \mathtt{ua} \notin \text{UA}_2, \mathtt{ua} \notin \text{UA}_{1_{\mathtt{des}}}, \mathtt{ua} \notin \text{UA}_1\}$
UA_4	$\{\mathtt{ua} \mid \mathtt{ua} \in \mathtt{des}(u_s) \wedge \mathtt{des}(\mathtt{ua}_t)\ \}$

in figure 3.1. UA_1 = {ua | ua = tail((ua$_p$, ars$_i$, ua$_w$)) $\vee$ ua$\in$anc(tail((ua$_p$, ars$_i$, ua$_w$))}

- UA_2: is a set of user attribute nodes on the path of the *source user* to a user attribute that contains both the *source user* and the *target policy element*. UA_2 = {ua | ua$\in$anc(head((ua$_p$, ars$_i$, ua$_w$))) $\wedge$ ua$\in$des(u_s)}

- UA_3: is a set of user attribute nodes that has a path to a user attribute containing both the *source user* and the *target policy element*, and excludes user attributes in the sets UA_1 and UA_2. UA_3 = {ua | ua$\in$anc(head((ua$_p$, ars$_i$, ua$_w$))), ua$\notin$ $UA_{2_{anc}}$, ua$\notin$ UA_2, ua$\notin$ $UA_{1_{des}}$, ua$\notin$ UA_1}

- UA_4: is a set of user attribute nodes that contains the *source user* and *target policy element* (u$_t$). UA_4 = {ua | ua$\in$des(u_s) $\wedge$ des(ua$_t$) }

The formal definition of the above queried *precondition association* and *authorization enabling policy element* sets for this scenario is in table 3.2.

Administrative Authorization Enabling Attributes

For an administrative request that the *source user* does not contain the *target policy element* or there is no attribute that contains the *source user* and the *target policy element*, the *target policy element* can be a user, user attribute, an object, or object attribute. To allow the *source user* operate on the *target policy element* in this scenario, we query for two *precondition associations* (*precondition association$'$* and *precondition association$''$*). While the *precondition association$''$* permits the *source user*'s operation on the *target policy element*, the *precondition association$'$* allows the creation of relation(s) that makes the predecessor node of the *precondition association$''$* reachable from the *source user*.

If the *target policy element* is an object or object attribute, *precondition association$'$* predecessor and successor nodes are user attributes. While the predecessor node of the *precondition association$''$* is a user attribute node, the successor is an object attribute, when the *target policy*

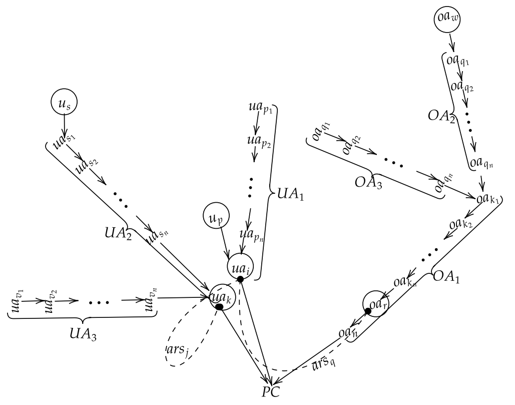


Figure 3.2: A Generic Authorization Graph: *source user* requesting administrative operation on an object attribute

element is an object or object attribute. However, if the *target policy element* is a user or user attribute, the predecessor and successor nodes of the *precondition association*$'$ and *precondition association*$''$ are user attributes. These *precondition associations* need to satisfy the following:

1. The successor node of the *precondition association*$''$ is reachable from the *target policy element* and a subset of its access rights set enables the *source user*'s operation.

2. The successor node of the *precondition association*$'$ is reachable from the predecessor node of the *precondition association*$''$.

3. There is path from a *privileged user* to the predecessor node of *precondition association*$'$ and *precondition association*$''$.

4. The successor node of the *precondition association*$'$ is reachable from the *source user* and the *privileged user*. A subset of the *precondition association*$'$ access rights set grants the *privileged user* the authority to create relation(s) that make(s) the predecessor node of the *precondition association*$''$ reachable from the *source user*.

In the authorization graph of figure 3.2, let u_s be the *source user* that wants to operate on an object attribute $\mathtt{oa}_w$. After the administrator queries for the two *precondition associations* (i.e., *precondition association*$'$ = ($\mathtt{ua}_i$, $\mathtt{ars}_j$, $\mathtt{ua}_k$) and *precondition association*$''$ = ($\mathtt{ua}_i$, $\mathtt{ars}_q$, $\mathtt{oa}_r$)in the figure). Then s/he queries for the following (*authorization enabling policy elements*) user attributes and object attributes sets in the authorization graph.

- UA_1 is a set of user attribute nodes that has a path to the predecessor-node(s) ($\texttt{ua}_i$) of both *precondition associations* ($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$) and ($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$). UA_1 = {ua | ua $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$))) $\wedge$ ua $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$)))}

- UA_2 is a set of user attribute nodes with a path to the successor-node ($\texttt{ua}_k$) of the *precondition association*$'$ ($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$) and are reachable from the *source user*. UA_2 = {ua | ua $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$))) $\wedge$ ua $\in$ des(u_s)}

- UA_3 is a set of user attribute nodes with a path to the successor-node of the *precondition association*$'$ ($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$) and not in the sets UA_1 and UA_2. UA_3 = {ua | ua $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_j$, $\texttt{ua}_k$))), ua $\notin$ $\mathrm{UA}_{2_{\text{anc}}}$, ua $\notin$ UA_2, ua $\notin$ $\mathrm{UA}_{1_{\text{des}}}$, ua $\notin$ UA_1}

- OA_1 is a set of object attribute nodes that has a path to the successor node ($\texttt{oa}_t$) of the *precondition association*$''$ ($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$) and is not reachable from the *target policy element* $\texttt{oa}_w$. OA_1 = {oa | oa $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$))) $\wedge$ oa $\notin$ des($\texttt{oa}_w$)$\wedge$ oa $\neq$ $\texttt{oa}_w$}

- OA_2 is a set of object attribute nodes that has a path to the successor node of the the *precondition association*$''$ ($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$) and is reachable from the the *target policy element* $\texttt{oa}_w$. OA_2 = {oa | (oa $\in$ des($\texttt{oa}_w$) $\wedge$ oa $\notin$ $\mathrm{OA}_{1_{\text{des}}}$) $\vee$ oa = $\texttt{oa}_w$}

- OA_3 is the set of object attributes that has a path to the successor node of the ($\texttt{oa}_t$) of the *precondition association*$''$ ($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$) and are reachable from the *target policy element* $\texttt{oa}_w$ and elements of the set OA_2. OA_3 = {oa | (oa $\in$ anc(head(($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$))) $\wedge$ oa $\in$ des($\texttt{oa}_w$) $\wedge$ oa $\in$ $OA_{2_{\text{des}}}$) $\vee$ oa = head(($\texttt{ua}_i$, $\texttt{ars}_q$, $\texttt{oa}_r$)) }

Table 3.3: Scenario-II Policy Elements & Relations.

Enabling Assoc-a	$\{(\mathtt{ua}_i, \mathtt{ars}_q, \mathtt{oa}_r) \in$ ASSOCIATION $\mid ar_x \subseteq \mathtt{ars}_q \land \exists\ \mathtt{u}_p \in$ U : $(\mathtt{u}_p, \mathtt{ua}_i) \in$ ASSIGN$^+\}$
Enabling Assoc-b	$\{(\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k) \in$ ASSOCIATION $\mid \exists\ \mathtt{u}_p, u_s \in$ U : $(u_s, \mathtt{ua}_i) \notin$ ASSIGN$^+ \land (\mathtt{u}_p, \mathtt{ua}_i) \in$ ASSIGN$^+\}$
Priv. Users	$\{\mathtt{u}_p \in$ U $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k), (\mathtt{ua}_i, \mathtt{ars}_q, \mathtt{oa}_r) \in$ ASSOCIATION : $(\mathtt{u}_p, \mathtt{ua}_p) \in$ ASSIGN$^+\}$
Source Users	$\{u_s \in$ U $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k) \in$ ASSOCIATION : $(u_s, \mathtt{ua}_k) \in$ ASSIGN$^+ \land (u_s, \mathtt{ua}_i) \notin$ ASSIGN$^+\}$
Target OA	$\{\mathtt{oa}_w \in$ OA $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_q, \mathtt{oa}_r) \in$ ASSOCIATION : $(\mathtt{oa}_w, \mathtt{oa}_r) \in$ ASSIGN$^+\}$
UA_1	$\{\mathtt{ua}_p \in$ UA $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k), (\mathtt{ua}_i, \mathtt{ars}_q, \mathtt{oa}_r) \in$ ASSOCIATION : $(\mathtt{ua}_p, \mathtt{ua}_i) \in$ ASSIGN$^+\}$
UA_2	$\{\mathtt{ua}_s \in$ UA $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k) \in$ ASSOCIATION, $u_s \in$ *Source Users* : $(u_s, \mathtt{ua}_s), (\mathtt{ua}_s, \mathtt{ua}_k) \in$ ASSIGN$^+\}$
UA_3	$\{\mathtt{ua}_v \in$ UA $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k) \in$ ASSOCIATION : $(\mathtt{ua}_v, \mathtt{ua}_k) \in$ ASSIGN$^+\} \setminus \{UA_1 \cup UA_2\}$
OA_1	$\{\mathtt{oa}_k \in$ OA $\mid \exists\ (\mathtt{ua}_i, \mathtt{ars}_q, \mathtt{oa}_r) \in$ ASSOCIATION, $\mathtt{oa}_q, \mathtt{oa}_t \in$ OA, $\mathtt{oa}_w \in$ *Target OA* : $(\mathtt{oa}_k, \mathtt{oa}_r), (\mathtt{oa}_w, \mathtt{oa}_t), (\mathtt{oa}_q, \mathtt{oa}_k), (\mathtt{oa}_t, \mathtt{oa}_k) \in$ ASSIGN$^+$ $(\mathtt{oa}_q, \mathtt{oa}_t), (\mathtt{oa}_t, \mathtt{oa}_q) \notin$ ASSIGN$^+\}$
OA_2	$\{\mathtt{oa}_t \in$ OA $\mid \mathtt{oa}_t \notin OA_1 \land \exists\ \mathtt{oa}_k \in OA_1$: $(\mathtt{oa}_t, \mathtt{oa}_k) \in$ ASSIGN$^+$
OA_3	$\{\mathtt{oa}_q \in$ OA $\mid \mathtt{oa}_q \notin OA_2 \land \exists\ \mathtt{oa}_k \in OA_1$: $(\mathtt{oa}_q, \mathtt{oa}_k) \in$ ASSIGN$^+$

Lastly, the administrator creates a combination of relation between elements of the *authorization enabling policy element*s using derived privileges from *precondition association*$'$ ($\mathtt{ua}_i$, $\mathtt{ars}_j$, $\mathtt{ua}_k$) and *precondition association*$''$ ($\mathtt{ua}_i$, $\mathtt{ars}_q$, $\mathtt{oa}_r$) to grant *source user* u_s request. Table 3.3 formally defines the above authorization enabling attributes and relations for a *source user* requesting an administrative operation on an object or an object attributes.

In a similar manner as the aforementioned procedure, an administrator can authorize a *source user* requesting an administrative operation on a user or user attribute. To represent this case in an authorization graph like figure 3.2, the object attribute sub-graph is a user attribute sub-graph. Also, all the five sets of *authorization enabling policy element*s are user attribute sets.

3.3 Illustrative Example

The example that follows demonstrates how the number of approaches to grant access can explode and how any constraint can limit the number of ways an access may be granted.

Example 1. *The authorization graph of figure 3.3 represents the access policy of a financial institution with the policy class BankOp Access. In the financial institution, a sensitive task 'trans-T' must be completed by a single user who is a member of both user attributes, ATM Custodian and Trans Serv Supervision (i.e., Cathy). The task 'trans-T' is modeled as a sequence of administrative operation granted on Object attributes, ATM & POS Serv and Wire Trans Serv through the sets of administrative access rights $aars_p$ and $aars_q$ respectively. Alice and Bob are employees of the financial institution. Alice is a member of ATM Custodian and not Trans Serv Supervision, while Bob is a member of Trans Serv Supervision and not ATM Custodian.*

For another task 'T-1' in this same institution, it is required that a user whose a member of both ATM Custodian and Trans Serv Supervision assigns a member of the Backup Officer to ATM Custodian in order to complete the task 'T-1'. In the current state of the authorization graph of figure 3.3, Cathy does not have the access right to assign Backup Officer to ATM Custodian. Assuming Jane and Paul (members of the institution that has a path to the user attribute Group Head) can grant Cathy the access right to create the required assignment (unlabeled edge), using *the administrative access rights in $aars_i$. The following are approaches to grant Cathy the access right to assign Backup Officer to ATM Custodian:*

1. Create association between user attribute: *Jane or Paul can create associations from any user attribute on Cathy's path to Op Officers (i.e., ATM Custodian and Trans Serv Supervision) to any user attribute that makes Op Officers reachable from Backup Officer (i.e., Backup Officer and Op Officers). In other words, Jane or Paul can create four different association relations (labeled with at least the access right Cathy is requesting) to grant Cathy the required access.*

2. Create assignment from a user attribute to user attribute: *Jane or Paul can create assignments (unlabeled edges) between user attributes that make Group Head or Regional Head reachable from Cathy. That is, in order to authorize Cathy's request, Jane or Paul needs to create an (unlabeled edge) assignment from user attribute node (ATM Custodian or Trans Serv Supervision) to user attribute node (Group Head or Regional Head). This second approach authorizes Cathy's request by creating one of the four possible assignment relations.*

3. Create assignment from a user to user attribute: *Again, Jane or Paul can create assignments (unlabeled edges) from user node (Cathy) to user attribute node (Group Head or Regional Head) that makes (Group Head or Regional Head) reachable from Cathy. There are only two ways to authorize Cathy's request using this approach.*

In total, there are twelve different ways of creating an assignment or association to allow *Cathy* complete task 'T-1'. However, only the two different ways of the three enumerated approaches does not violate the constraint on the task 'trans-T'. If *Jane* or *Paul* should authorize *Cathy*'s request using the first or the second approach, then *Alice* and *Bob* can collude to carry out task 'trans-T'.

Even more, this is a simple example compare to the size of an enterprise where this kind of issue get more complicated. Also, the structure of an authorization graph may permit granting an access using any non-redundant combination of the three approaches. For instance, in the previous example, *Jane* or *Paul* may grant *Cathy*'s request to complete task 'T-1' by first creating

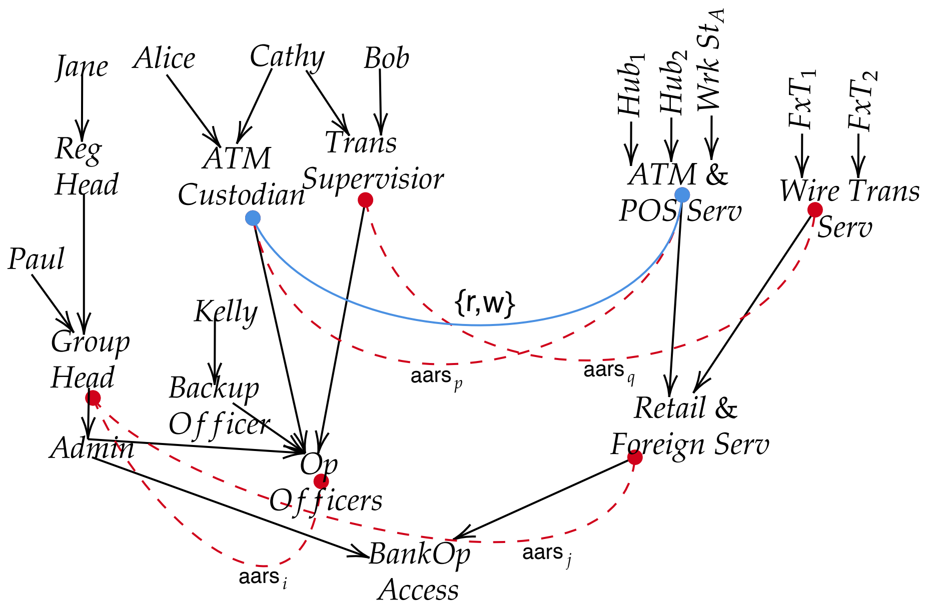


Figure 3.3: Policy Machine Authorization Graph

an (unlabeled edge) assignment from *Cathy* to *Backup Officer* and then create another (unlabeled edge) assignment from *Backup Officer* to *Group Head* or *Regional Head*.

3.4 Algorithmic Design and Evaluation

3.4.1 Policy Review Algorithm for Generic Authorization

The proposed algorithm in this dissertation comprises of three functions. The function `mainPReview` is the basis for the discovery of policy elements and relations for the two scenarios. The functions `targetContainsSource` and `sourceNotContained` provide possible approaches to authorize a *source user* requesting access in scenario-I, and scenario-II respectively.

The *privileged user*'s access rights are utilized to create the authorization approaches. All administrative access rights for creating assignments enable a common operation, `createAssign`. The `createAssign` takes an ordered pair of nodes that an edge originates and terminates as input. Similarly, access rights for creating associations enable the operation `createAssoc`. A triple as input to the `createAssoc` is the predecessor-node, the label, and successor-node of an edge.

An input to the mainPReview function is the authorization state of a graph G$_{auth}$ and access requesting triple `areq`$_d$. The p_{u_s}, `ar`$_x$, and `target` of the triple is the *source user*, access right *source user* is requesting, and *target policy element* respectively. The expected output of this algorithm $AREQ_A$ is a list of sets of possible approaches to grant the *source user*'s access for either of two scenarios.

The flow of the algorithm is as follows:

1. Step 1a of algorithm 3.1 are preconditions to generate authorization approaches for scenario-I and scenario-II, respectively. If the conditions in the previous lines were satisfied, the function

   `scenario-IPEGenerator` generates the sets $prev.Users$ (PU), UA_1, UA_2, UA_3, and, UA_4 as defined in table 3.2. The sets of policy elements returned from the function `scenario-IPEGenerator`, a set ARS$_{assoc}$ of `ars`$_j$ subsets that include access the *source user* is requesting, an empty list $AREQ_A$ and the *source user* `areq`$_d$$\cdot p_{u_s}$ are augments of the function `targetContainsSource`.

   (a) The function `targetContainsSource` loops through the powerset of access rights available to the *privileged user*.

   (b) Within the loop, each if statement appends the set(s) of triple that can create edges to authorize *source user*'s access request.

   (c) If the elements of the powerset are exhausted, then $AREQ_A$ returns the list of sets of approaches to authorize the *source user*'s request.

2. If the `target` is not a user or user attribute, then only scenario-II preconditions are true. Access requesting triple `areq`$_d$ that satisfies scenario-II preconditions, step (1c) of algorithm 3.1 and authorization graph G$_{auth}$ are augments of the function `scenario-IIPEGenerator`. Table 3.3 defines the sets of policy elements, $prev.Users$ (PU), UA_1, UA_2, UA_3, OA_1, OA_2, and OA_3, the function `scenario-IIPEGenerator` generates. These sets returned

by scenario-IIPEGenerator, a set $\mathtt{ARS}_{assoc}$ of $\mathtt{ars}_q$ subsets that include access the *source user* is requesting, an empty list $AREQ_A$ and the triple $\mathtt{areq}_d{\cdot}p_{u_s}$ are augments of the function sourceNotContained. The flow of the function sourceNotContained is the same as the previous scenario.

Algorithm 3.1 Main function for the algorithm to generate approaches to authorize access request

```
Input: $\mathtt{areq}_d = (p_{u_s}, \mathtt{ar}_x, \mathtt{target})$, $\mathbf{G}_{auth}$ = (PE, ASSIGN, ASSOCIATION)
Output: $AREQ_A$
Step 1: /*Group administrative and resource Associations */
  a. $AREQ_A \leftarrow [\ ]$
  b. If $\mathtt{areq}_d{\cdot}\mathtt{target} \in \{PE{\cdot}U \cup PE{\cdot}UA\}$
      i. If $\exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k) \in G_{auth}$.ASSOCIATION : $(\mathtt{areq}_d.user, \mathtt{areq}_d{\cdot}\mathtt{target}) \in$
         $\mathrm{ASSIGN}^+ \wedge (\mathtt{areq}_d.user, \mathtt{ua}_k) \in \mathrm{ASSIGN}^+ \wedge (\mathtt{areq}_d{\cdot}\mathtt{target}, \mathtt{ua}_k) \in \mathrm{ASSIGN}^+$
         $\wedge \{\mathtt{areq}_d{\cdot}\mathtt{ar}_x\} \subseteq \mathtt{ars}_j$
           $\mathtt{PU}, \mathtt{UA}_1, \mathtt{UA}_2, \mathtt{UA}_3, \mathtt{UA}_4 \leftarrow$ scenario-IPEGenerator$\mathtt{areq}_d$, $G_{auth}$, $(\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k)$
           Let $\mathtt{ARS}_{assoc} \leftarrow \{ars : ars \in 2^{\mathtt{ars}_j} \wedge \mathtt{areq}_d{\cdot}\mathtt{ar}_x \in ars\}$
           Return $\mathtt{areq}_d{\cdot}p_{u_s}$, $AREQ_A$, $\mathtt{UA}_1$, $\mathtt{UA}_2$, $\mathtt{UA}_3$, $\mathtt{UA}_4$, PU, $\mathtt{ARS}_{assoc}$, $\mathtt{ars}_j$
  c. ElseIf $(\mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{areq}_d{\cdot}\mathtt{target}) \notin \mathrm{ASSIGN}^+ \wedge \exists\ (\mathtt{ua}_i, \mathtt{ars}_j, \mathtt{ua}_k), (\mathtt{ua}_p, \mathtt{ars}_q, \mathtt{oa}_r)$
     $\in G_{auth}$.ASSOCIATION : $(\mathtt{areq}_d{\cdot}\mathtt{target}, \mathtt{oa}_r) \in \mathrm{ASSIGN}^+ \wedge (\mathtt{ua}_p, \mathtt{ua}_i) \in \mathrm{ASSIGN}^+$
     $\wedge (\mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_k) \in \mathrm{ASSIGN}^+ \wedge (\mathtt{ua}_i, \mathtt{ua}_k) \in \mathrm{ASSIGN}^+ \wedge \{\mathtt{areq}_d{\cdot}\mathtt{ar}_x\} \in \mathtt{ars}_q$
       Let $\mathtt{ARS}_{assoc} \leftarrow \{ars : ars \in 2^{\mathtt{ars}_q} \wedge \mathtt{areq}_d{\cdot}\mathtt{ar}_x \in ars\}$
       Return sourceNotContained($\mathtt{areq}_d{\cdot}p_{u_s}$, $AREQ_A$, $\mathtt{UA}_1$, $\mathtt{UA}_2$, $\mathtt{UA}_3$, $\mathtt{OA}_1$, $\mathtt{OA}_2$, $\mathtt{OA}_3$, PU,
       $\mathtt{ARS}_{assoc}$, $\mathtt{ars}_j$, $\mathtt{ars}_q$)
```

3.4.2 Performance Evaluation of Access Authorization Review Using Networkx

We discuss the implementation details of the generic access authorization review algorithm (see 3.4.1) utilizing the networkx (python library for studying graphs and networks). An Ubuntu virtual machine with two cores and 10Gb of memory was used as our experimental platform. We tested the response time of our algorithm to return all the approaches that authorize requests by simulating 10 random graphs for scenario-II. For each of the random graphs, we performed 200 iterations for each request sizes of 1, 5, 500, and 1000. The 10 random access control graphs have 1000 (user,

Algorithm 3.2 A function to generate approaches to authorize a *source user* access request for scenario-I

function targetContainsSource($\texttt{areq}_d$, $AREQ_A$, $\texttt{UA}_1$, $\texttt{UA}_2$, $\texttt{UA}_3$, $\texttt{UA}_4$, $\texttt{PU}$, $\texttt{ARS}_{assoc}$, $\texttt{ars}_j$)

```
1. Let $\texttt{ua}_i \in \texttt{UA}_i$, $p_{u_p} \in \texttt{PU}$, and $\texttt{ars}_{assoc} \in \texttt{ARS}_{assoc}$
2. For $\texttt{ars} \in 2^{\texttt{ars}_j}$
    a. If $\{\{\texttt{c-uua}\}\} == \texttt{ars}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{areq}_d{\cdot}p_{u_s}, \texttt{ua}_1\rangle)\Big\}$
    b. If $\{\ \{\texttt{c-uaua}\}\} == \texttt{ars}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{ua}_2, \texttt{ua}_1\rangle)\Big\}$
    c. If $\{\{\texttt{c-uua}\},\ \{\texttt{c-uaua}\}\} == \texttt{ars}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{areq}_d{\cdot}p_{u_s}, \texttt{ua}_3\rangle), (p_{u_p}, \texttt{createAssign}, \langle \texttt{ua}_3, \texttt{ua}_1\rangle)\Big\}$
    d. If $\{\{\texttt{c-assoc-fr-ua}\}, \{\texttt{c-assoc-to-ua}\}\} == \texttt{ars}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssoc}, \langle \texttt{ua}_2, \texttt{ars}_{assoc}, \texttt{ua}_4\rangle)\Big\}$
    e. If $\{\{\texttt{c-uua}\}, \{\texttt{c-assoc-fr-ua}\}, \{\texttt{c-assoc-to-ua}\}\}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{areq}_d{\cdot}p_{u_s}, \texttt{ua}_3\rangle), (p_{u_p}, \texttt{createAssoc}, \langle \texttt{ua}_3, \texttt{ars}_{assoc}, \texttt{ua}_4\rangle)\Big\}$
    f. If $\{\ \{\texttt{c-uaua}\}, \{\texttt{c-assoc-fr-ua}\}, \{\texttt{c-assoc-to-ua}\}\} == \texttt{ars}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{ua}_2, \texttt{ua}_3\rangle), (p_{u_p}, \texttt{createAssoc}, \langle \texttt{ua}_3, \texttt{ars}_{assoc}, \texttt{ua}_4\rangle)\Big\}$
    g. If $\{\texttt{c-uua}\},\ \{\texttt{c-uaua}\}, \{\texttt{c-assoc-fr-ua}\}, \{\texttt{c-assoc-to-ua}\}$
           $AREQ_A = AREQ_A \cup \Big\{ (p_{u_p}, \texttt{createAssign}, \langle \texttt{areq}_d{\cdot}p_{u_s}, \texttt{ua}_3\rangle), (p_{u_p}, \texttt{createAssign}, \langle \texttt{ua}_1, \texttt{ua}_3\rangle), (p_{u_p}, \texttt{createAssoc}, \langle \texttt{ua}_3, \texttt{ars}_{assoc}, \texttt{ua}_4\rangle)\Big\}$
3. Return $AREQ_A$
```

Algorithm 3.3 A function to generate approaches to authorize a *source user* access request for scenario-II

```
function sourceNotContained($\mathtt{areq}_d$, $AREQ_A$, $\mathtt{UA}_1$, $\mathtt{UA}_2$, $\mathtt{UA}_3$, $\mathtt{OA}_1$, $\mathtt{OA}_2$, $\mathtt{OA}_3$, PU, $\mathtt{ARS}_{assoc}$, $\mathtt{ars}_j$, $\mathtt{ars}_q$)
  1. Let $\mathtt{ua}_i \in \mathtt{UA}_i$, $\mathtt{oa}_i \in \mathtt{OA}_i$, $p_{u_p} \in \mathtt{PU}$, and $\mathtt{ars}_{assoc} \in \mathtt{ARS}_{assoc}$
  2. For $\mathtt{ars} \in 2^{\{\mathtt{ars}_j \cup \mathtt{ars}_q\}}$
      a. If {{c-uua}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_1 \rangle)\}$
      b. If { {c-uaua}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{ua}_2, \mathtt{ua}_1 \rangle)\}$
      c. If {{c-assoc-fr-ua}, {c-assoc-to-oa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssoc, $\langle \mathtt{ua}_2, \mathtt{ars}_{assoc}, \mathtt{oa}_1 \rangle)\}$
      d. If {{c-uua}, {c-uaua}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssign, $\langle \mathtt{ua}_3, \mathtt{ua}_1 \rangle)\}$
      e. If {{c-uua}, {c-assoc-fr-ua}, {c-assoc-to-oa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssoc, $\langle \mathtt{ua}_3, \mathtt{ars}_{assoc}, \mathtt{oa}_1 \rangle)\}$
      f. If { {c-uaua}, {c-assoc-fr-ua}, {c-assoc-to-oa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{ua}_2, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssoc, $\langle \mathtt{ua}_3, \mathtt{ars}_{assoc}, \mathtt{oa}_1 \rangle)\}$
      g. If {{c-assoc-fr-ua}, {c-assoc-to-oa}, {c-oaoa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssoc, $\langle \mathtt{ua}_2, \mathtt{ars}_{assoc}, \mathtt{oa}_3 \rangle)$, $(p_{u_p}$, createAssign, $\langle \mathtt{oa}_2, \mathtt{oa}_3 \rangle)\}$
      h. If {{c-uua}, {c-uaua}, {c-assoc-fr-ua}, {c-assoc-to-oa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssign, $\langle \mathtt{ua}_2, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssoc, $\langle \mathtt{ua}_3, \mathtt{ars}_{assoc}, \mathtt{oa}_1 \rangle)\}$
      i. If {{c-uua}, {c-assoc-fr-ua}, {c-assoc-to-oa}, {c-oaoa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{areq}_d{\cdot}p_{u_s}, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssoc, $\langle \mathtt{ua}_3, \mathtt{ars}_{assoc}, \mathtt{oa}_3 \rangle)$, $(p_{u_p}$, createAssign, $\langle \mathtt{oa}_2, \mathtt{oa}_3 \rangle)\}$
      j. If { {c-uaua}, {c-assoc-fr-ua}, {c-assoc-to-oa}, {c-oaoa}} == ars
          $AREQ_A = AREQ_A \cup \{ (p_{u_p}$, createAssign, $\langle \mathtt{ua}_2, \mathtt{ua}_3 \rangle)$, $(p_{u_p}$, createAssoc, $\langle \mathtt{ua}_3, \mathtt{ars}_{assoc}, \mathtt{oa}_3 \rangle)$, $(p_{u_p}$, createAssign, $\langle \mathtt{oa}_2, \mathtt{oa}_3 \rangle)\}$
```

Scenario-I Node Proportion	
User Nodes	200
User Attribute Nodes	800
Policy Class Node	1
Scenario-II Node Proportion	
User Nodes	200
Object Nodes	200
User Attribute Nodes	300
Object Attribute Nodes	300
Policy Class	1

Table 3.4: Proportion of Nodes for Scenario-I and Scenario-II

user attribute, object, and object attribute) nodes. The proportion of nodes of each type per access control graph is shown in the table 3.4. We restricted the maximum number of edges from a user to the policy class node to 5. Figure 3.4 is the distribution of response time to generate all possible approaches to grant a request. Again, the NIST policy machine reference documentation and no related work in the literature addressed this policy query of our work, no comparative experiment for us to perform.

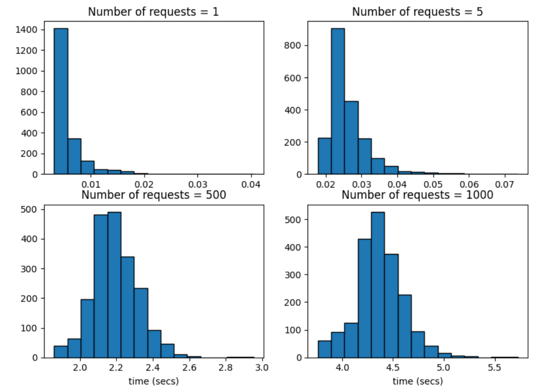


Figure 3.4: Distribution of Response Time to Generate Access Authorization Approaches for Scenario-II on graphs with 1,000 nodes

CHAPTER 4: CONSTRAINED ACCESS AUTHORIZATION AND REVOCATION THROUGH POLICY REVIEW

This chapter expands and improves on the work of the previous chapter. We introduce parameters that make it possible to limit the granted authorization. Further, this version of the algorithm can query for approaches to revoke access. The objective of the work in this chapter is to answer two questions:

i. If a *user* is allowed to perform *op* on *resource*, What are the approaches to deny the *user* access to perform *op* on the protected *resource*?

ii. If a *user* is not authorized to perform *op* operation on a *resource*, What are the approaches to grant the *op* on protected *resource* to the *user*?

4.1 Policy Review For Constrained Access Authorization

In the previous (generic) algorithm, relation(s) are created from the access enablers sets to allow the *source user* perform requested access. In a real-world application, the size of the policy authorization graph is big. The generated approaches to authorize a request tend to become a deluge of information for an administrator to process. Using the illustrative example of chapter 3, where one of the approaches to authorize *Cathy*'s request was to create a user attribute assignment. From the authorization graph in reference and the definition of user attribute enablers, the sets $\texttt{UA}_1$ = {*Group Head*, *Regional Head*} and $\texttt{UA}_2$ = {*ATM Custodian*, *Trans Serv Supervision*}. The number of possible approaches to authorize *Cathy*'s request using the user attribute assignment is the cartesian product of the two sets. Thus, the number of possible ways of granting a request through a given set of relations is proportional to the cardinality (size) of access authorization enablers sets.

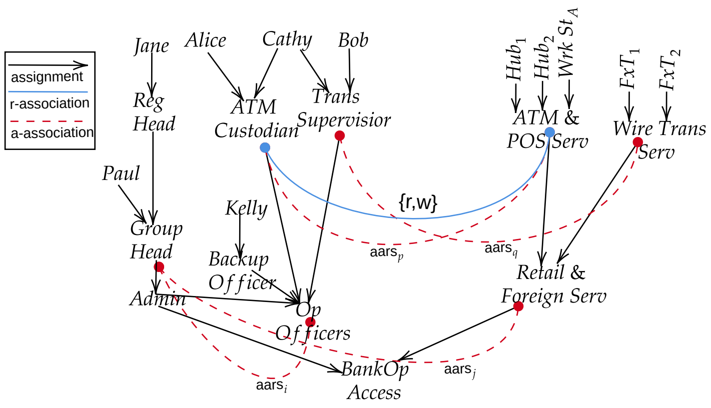


Figure 4.1: Policy Machine Authorization Graph

4.1.1 Derived functions

We categorize a request for constraint authorization or revocation into two scenarios as before. In the first scenario, the *target policy element* contains the *source user* or a user attribute contains *source user*and *target policy element*. This scenario deals with the case of a constrained authorization or revocation on a request to operate on user or user attribute *target policy element*. In the other case, no containment relationship between the *source user* and *target policy element*. We query for constraint authorization or revocation approaches for requests on any *target policy element* except the policy class. Some of the *authorization enabling policy element* definitions are the same as in chapter 3, while others are derivates. The functions that are the building blocks for the definition of the *authorization enabling policy element* in the previous work stay the same. For easy reference, below are the function definition:

- `tail` : ASSOCIATION $\longrightarrow$ `UA`: is a function that maps an edge, association relation, $(ua_i, ars_j, au_k) \in$ ASSOCIATION to the (user attribute) node $ua_i \in$ `UA` it originates.
- `head` : ASSOCIATION $\longrightarrow$ `AT`: is a function that maps an edge, association relation, $(ua_i, ars_j, au_k) \in$ ASSOCIATION to the (user/object attribute) node $at_k \in$ `AT` it terminates. Where `AT` = `UA` $\cup$ `OA`
- `anc`: `PE` $\longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is

an ancestor to the policy element.

- $\texttt{des}\colon \texttt{PE} \longrightarrow 2^{\texttt{PE}}$: is the mapping from a policy element to the set of policy elements that is a descendant to the policy element.

- $\texttt{PE}_{i_{\texttt{func}}} = \{\texttt{node} \mid (\exists pe_j \in \texttt{PE}_i)[\texttt{node} \in \texttt{func}(pe_j)]\}$: is the set of all policy elements returned by `func` for the set $\texttt{PE}_i$, where `func` is `anc` or `des`.

4.1.2 Constrained Access Authorization Methodologies

The results of the authorization review from the previous chapter provide an administrator the ability to know all the relations any user can create to permit another users' request. A policy machine administrator must have the overall answer to a given policy review query. However, apart from the possibility that the policy review for user authorization returns a flood of information, we may be interested in only a subset of the possible approaches to grant requested access. For instance, the policy administrator of *BankOp Access* may be interested in knowing how to authorize *Cathy*'s request without using the access-enabling set $\texttt{UA}_2$. Rather than generate the entire possible means to allow access, as in the previous algorithmic design, we introduced additional input parameters for the algorithm to adapt to tailored queries.

Before discussing the added parameters to the previous algorithm for achieving flexibility, there is a need to know the scope of allowed access for a given approach in a policy review of authorization. We developed constraints based on the knowledge of the extent of granted access for any authorization approach. There is a discernible sequence in the relations created to authorize the *source user* request. In other words, we observe a pattern in the direction of edges when the algorithm generates authorization approaches. These are edges created from or to some authorization enabling attribute sets that allow the *source user* perform requested operation on the *target policy element*. That is, relations (edges) start from a tail node and end at the head node. The scope of an authorization approach is limited to the elements of an attribute set that are tail nodes for edges (relations) in an authorization approach.

Table 4.1: Scope of Authorization/Revocation on Attribute Groups

Attribute Groups	**Pattern of Relations(s)**	**Authorization Effect**
UA_1	Assignment: to	No effect
UA_2	Assignment: from Association: from	Access granted or inherited
UA_3	Assignment: from and to Association: from	Access granted or inherited
UA_4	Association: to	No effect
OA_1	Association: to	No effect
OA_2	Association: to Assignment: from	No effect Access entry granted
OA_3	Association: to	No effect

The creation of the relation(s) that authorizes the *source user* grants capabilities or access entries for all the possible tail nodes (authorization enabling attributes) in a relation(s) that allow(s) access for a requested user. Using the earlier example of a user attribute assignment approach to authorizing *Cathy*'s request, the elements of the sets $\mathtt{UA}_2$ and $\mathtt{UA}_1$ are the tail and head nodes, respectively. In figure, before the creation of the user attribute assignment, the granted capabilities set of the elements *ATM Custodian* and *Trans Serv Supervision* nodes of set $\mathtt{UA}_2$ are $\big\{(aars_p$, *ATM & POS Serv*), ({ r,w}, *ATM & POS Serv*)$\big\}$ and $\big\{(aars_q$, *Wire Trans Serv*)$\big\}$, respectively. Assuming access granted to *Cathy* was through the creation of user attribute assignment from *Trans Serv Supervision* to *Regional Head* node. The assigned and acquired capabilities set of *Trans Serv Supervision* is now $\big\{(aars_q$, *Wire Trans Serv*), $(aars_i$, *Op Officers*), $(aars_j$, *Retail & Foreign Serv*)$\big\}$.

In general, the authorization enabling attributes $\mathtt{UA}_2$, $\mathtt{UA}_3$, and $\mathtt{OA}_2$ are the sets with tail nodes in edges that grant requested access. When an edge (assignment or association relation) from $\mathtt{UA}_2$ and $\mathtt{UA}_3$ nodes authorizes a *source user*'s request, there are added capabilities for these user attribute sets. The creation of an assignment relation from elements of the set $\mathtt{OA}_2$ that allows access to the target policy element elevates the access entries of the set $\mathtt{OA}_2$ elements. The table summarizes the pattern of relation(s) created using these attribute sets and the change in capability or access entry of the attributes sets after an authorization.

The column pattern of relation created (i.e., <relation/edge type> : <direction>) in the table describe the type of edge(s) we can create from or to elements of a given access enabling attribute set. As an example when the resource is a user or user attribute, a possible approach to authorize access is creating an edge (assignment) from $\mathtt{ua}_2$ to $\mathtt{ua}_1$ or creating an assignment from $\mathtt{ua}_2$ to $\mathtt{ua}_3$ and creating an association from $\mathtt{ua}_3$ to $\mathtt{ua}_4$, where $\mathtt{ua}_i \in UA_i$. The third column of the table signifies the change in capability or access entry of a user or an object attribute, respectively, after utilizing an authorization approach. Authorizing a request to elevate the capability of the user attributes UA_2 and UA_3, and access entry of object attribute OA_2. A policy administrator can constrain the authorization of a request through these attributes with elevated capability or access entry.

In addition to request (*user*, *op*, *resource*), and a graph associated with the request as input parameters in the previous algorithm, we introduced a record (*authmode*) with fields of key-value pair. Firstly, if there is an association relation (policy) that grants the user the authority to perform *op* on the *resource*, the algorithm generates approaches to revoke the access. Otherwise, it produces possible relation(s) that allow *user* to perform *op* on the *resource*. The key-value pairs from the input record allow a policy administrator to specify modes of authorization. The algorithm can generate all possible approaches with/without constraint to authorize a request. A key *isGeneric* with a boolean value of true generates all approaches without restriction, while the value of false produces constrained authorization approaches. When *isGeneric* is true the two other key-value pair in the record becomes null. Another key is the *denySet*, and the value is a set that authorization granted or inherited by its elements is constrained. The value for the *denySet* can take take $\mathtt{ua}_2$, UA_3, or OA_2 attribute set. The above definition of the access enabling sets have no constraint. The size of set $\mathtt{UA}_2$, $\mathtt{UA}_3$, and $\mathtt{OA}_2$ becomes smaller or empty when utilizing the discussed constraining parameters.

For example, if the key *denySet* has a value UA_2, the algorithm excludes all relations(s) that authorize access through this user attribute set. An attribute set with elevated capability or access entry that is not the value of the *denySet* key is also constrained through the third key *limitto*. The value for the key *limitto* specifies the number of elements used to generate approaches to authorize access. Its value is a user attribute set if the resource is a user or user attribute and an object attribute set for an object or object attribute resource. Assuming a request = (*user*, *op*, *resource*), Graph = (PE, ASSIGN, ASSOCIATION), *authmode* = {*isGeneric* : false, *denySet* : UA_2, *limitto* : 1}, these input parameters yields an output of constrained authorization approaches. It excludes authorization approaches using elements in the user attribute set UA_2. The value of *limitto* permits creating authorization approaches using one element of UA_3 or OA_2 if the resource is a user or an object type, respective.

4.1.3 Constrained Access Enablers

The following defined sets of (user and object) attribute groups form the basis of our algorithm for the policy review of access authorization and revocation. We derived the attribute groups considering the resource a user wants to perform an action on is of type user or object. When the resource in question is a user or user attribute, the following user attribute groups create relations that authorize and revoke access requests.

- UA_1 = {ua | ua = tail((ua_i, ars_j, au_k)) $\vee$ ua$\in$anc(tail((ua_i, ars_j, au_k)))}
- UA_2 = {ua | ua$\in$anc(head((ua_i, ars_j, au_k))) $\wedge$ ua$\in$des(*user*)}
- UA_3 = {ua | ua$\in$anc(head((ua_i, ars_j, au_k))), ua$\notin$ $UA_{2_{anc}}$, ua$\notin$ UA_2, ua$\notin$ $UA_{1_{des}}$, ua$\notin$ UA_1}
- UA_4 = {ua | ua$\in$des(*user*) $\wedge$ des(*resource*) }

Where (ua_i, ars_j, au_k) is an association relation that grants the user attribute ua_i the access rights ars_j on au_k.

Assuming we want to grant or deny access to an object or object attribute. Combining the sets UA_1, UA_2, UA_3, above and the following object attribute groups enable the creation of relations that authorize or revoke access.

- OA_1 = $\{oa \mid oa \in \texttt{anc}(\texttt{head}((ua_p, ars_q, ao_r))), oa \notin \texttt{des}(\textit{resource})$,oa $\neq$ *resource*$\}$
- OA_2 = $\{oa \mid (oa \in \texttt{des}(\textit{resource}) \wedge oa \notin \mathrm{OA1}_{\texttt{des}}) \vee oa = \textit{resource}\}$
- OA_3 = $\{oa \mid (oa \in \texttt{anc}(\texttt{head}((ua_p, ars_q, ao_r))), oa \in \texttt{des}(\textit{resource}), oa \in OA_{2_{\texttt{des}}}\ ) \vee oa = \texttt{head}((ua_p, ars_q, ao_r))\ \}$

This scenario requires two association relations. The association (ua_i, ars_j, au_k) grants authority to create or delete the relation(s) from attribute(s) of the user to whom we want to authorize/deny access. The second association (ua_p, ars_q, ao_r) allows the creation or deletion of relation(s) to the requested resource (object or object attribute).

4.2 Policy Review For Access Revocation

In the case of policy queries to revoke access, the *privileged user* has the authority to disassociate the *source user* from the granted authorities that enable the previous access granted to the *target policy element.* By deleting a combination of relations, user to user attribute, user attribute to user attribute, association, object to object attribute, and object attribute to object attribute, revokes the *source user*'s access. The assumption is that none of these delete operation require the removal of the *source user*, *target policy element*, or any attribute to retain a valid state of the policy graph. Rather than query for *authorization enabling policy element* and *precondition associations*, we query for *precondition associations* and what we term as *revocation relations*. The *revocation relations* are the ordered pair of nodes (policy elements) that the removal of an edge between them disassociates authorities granted to the *source user.*

The access revocation scenarios are the opposite of the access authorization scenarios. Thus, the are three access revocation scenarios, the administrative revocation by user attribute, administrative revocation of attribute, and non-administrative revocation by object attributes. This is the scenario where the *source user* and *target policy element* are in the same sub-graph. The combination of delete user to user attribute, user attribute to user attribute, association relations revokes the *source user*'s access to the *target policy element.*

For this scenario, the *source user* and *target policy element* are not in the same sub-graph, the *target policy element* is any of the possible types (i.e., user, user attribute, object, or object attribute). If the *target policy element* is a user or user attribute, the *revocation relations* are user to user attribute, user attribute to user attribute relations, and association relations. Likewise, for an object or object attribute *target policy element*, the *revocation relations* are object to object attribute, object attribute to object attribute, and association relations.

4.3 Illustrative Example

4.3.1 Policy Review for Scenario-I

This first example elaborates on the illustrative example of chapter 3 and compares the policy review for authorization of *Cathy*'s request using the generic and constrained algorithms. We provide the access enabling attribute sets using the derived function definition on the policy authorization graph of figure 4.1. Rather than the single operation approaches of authorization *Cathy*' s request used in the example 3.3, this example provides all the methods using the combination of the three operations for this scenario - creation of user assignment, user attribute assignment, and association relations(edges). The multiple operations that create user assignment, user attribute assignment, user assignment, association, user attribute assignment, association, and user assignment, user attribute assignment, association sets of relations also authorize the request for this scenario.

The creation of a relation(an edge) using user assignment, user attribute assignment, and association operation from *Cathy* to user attributes in the set $\mathtt{UA}_1$, from user attributes in $\mathtt{UA}_2$ to $\mathtt{UA}_3$, and from $\mathtt{UA}_2$ to $\mathtt{UA}_4$, respectively, are the single operation approaches to allow *Cathy* to assign *Backup Officer* to *Trans Serv Supervision.* A double operation approaches to policy review for authorization is using the sets of two relations, user assignment, user attribute assignment, user assignment, association, and user attribute assignment, association, to create sets of relations from *Cathy* to $\mathtt{UA}_3$, from $\mathtt{UA}_3$ to $\mathtt{UA}_1$, from *Cathy* to $\mathtt{UA}_3$, from $\mathtt{UA}_3$ to $\mathtt{UA}_4$, and from $\mathtt{UA}_2$ to $\mathtt{UA}_3$, from $\mathtt{UA}_3$ to $\mathtt{UA}_4$, respectively. Lastly, the set of relations(edges), from *Cathy* to set $\mathtt{UA}_3$ user attributes, from the set $\mathtt{UA}_1$ to $\mathtt{UA}_3$ user attributes, from $\mathtt{UA}_3$ to $\mathtt{UA}_4$ user attributes , applies the operations that create the user assignment, user attribute assignment, and association relations.

Table 4.2: Generic access authorization approaches for *Cathy*'s request

- User Assignment

  $\{$(*Cathy*, *Group Head*), (*Cathy*, *Regional Head*)$\}$

- User Attribute Assignment

  $\{$(*ATM Custodian*, *Group Head*), (*ATM Custodian*, *Regional Head*), (*Trans Serv Supervision*, *Group Head*), (*Trans Serv Supervision*, *Regional Head*)$\}$

- Association from User Attribute to User Attribute

  $\{$(*ATM Custodian*, $aars_k$, *Op Officers*), (*Trans Serv Supervision*,$aars_k$, *Op Officers*)$\}$

- User Assignment and User Attribute Assignment

  $\{\{$(*Cathy*, *Backup Officer*), (*Backup Officer*, *Group Head*)$\}$, $\{$(*Cathy*, *Backup Officer*), (*Backup Officer*, *Regional Head*)$\}\}$

- User Assignment and Association from User Attribute to User Attribute

  $\{\{$(*Cathy*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)$\}\}$

- User Attribute Assignment and Association from User Attribute to User Attribute

  $\{\{$(*ATM Custodian*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)$\}$, $\{$(*Trans Serv Supervision*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)$\}\}$

- User Assignment, User Attribute Assignment, and Association from User Attribute to User Attribute

  $\{\{$(*Cathy*, *Backup Officer*), (*Group Head*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)$\}$, $\{$(*Cathy*, *Backup Officer*), (*Regional Head*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)$\}\}$

Table 4.3: Constrained access authorization approaches for *Cathy*'s request, without set $\texttt{UA}_3$

– User Assignment

$\big\{$(*Cathy*, *Group Head*), (*Cathy*, *Regional Head*)$\big\}$

– User Attribute Assignment

$\big\{$(*ATM Custodian*, *Group Head*), (*ATM Custodian*, *Regional Head*), (*Trans Serv Supervision*, *Group Head*), (*Trans Serv Supervision*, *Regional Head*)$\big\}$

– Association from User Attribute to User Attribute

$\big\{$(*ATM Custodian*, $aars_k$, *Op Officers*), (*Trans Serv Supervision*,$aars_k$, *Op Officers*)$\big\}$

Table 4.4: Constrained access authorization approaches for *Cathy*'s request, without set $\texttt{UA}_2$

– User Assignment

$\big\{$(*Cathy*, *Group Head*), (*Cathy*, *Regional Head*)$\big\}$

– User Assignment and User Attribute Assignment

$\big\{${(*Cathy*, *Backup Officer*), (*Backup Officer*, *Group Head*)}, {(*Cathy*, *Backup Officer*), (*Backup Officer*, *Regional Head*)}$\big\}$

– User Assignment and Association from User Attribute to User Attribute

$\big\{${(*Cathy*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)}$\big\}$

– User Assignment, User Attribute Assignment, and Association from User Attribute to User Attribute

$\big\{${(*Cathy*, *Backup Officer*), (*Group Head*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)}, {(*Cathy*, *Backup Officer*), (*Regional Head*, *Backup Officer*), (*Backup Officer*, $aars_k$, *Op Officers*)}$\big\}$

The three listings below provide the result for operations that authorizes Cathy's request. Table 4.2 is the review of authorization for Cathy's request without constraint. Tables 4.3 and 4.4 limit the sets $\texttt{UA}_2$ to 1 and $\texttt{UA}_3$ to 0, and $\texttt{UA}_2$ to 0 and $\texttt{UA}_3$ to 1, respectively.

4.3.2 Policy Review for Scenario-II

This second example demonstrates the use of our algorithm for a policy review of authorization on an access request that we described as the second scenario (scenario-II). From the authorization graph of the financial institution (see figure 4.1), the object attribute *Retail & Foreign Serv* is the root folder for the other folders (object attributes, i.e., *Wire Trans Serv*, *ATM & POS Serv*, Hub_1, Hub_2, FxT_1, FxT_2) and the file $Wrk\ St_A$. No *Trans Serv Supervision* (i.e., *Cathy* and *Bob*) has an administrative right on the folder (object attribute *Wire Trans Serv*) and the folder it contains.

Let the administrative access right set $aars_p$ grants *ATM Custodian* to create files and folders in the and sub-folders. However, *Bob*, a *Trans Serv Supervision*, needs the access rights to create and delete files and folders in Hub_1. Since the *target policy element* is an object attribute, a query for approaches to authorization *Bob*'s request requires user and object attribute access enabling sets. In scenario-I illustrated by the previous example, three single operations can create a relation for authorizing a request. The only single relation in this scenario that can grant access is the association relation.

The access enabling sets for the association relation are $\mathtt{UA}_2$ and $\mathtt{OA}_3$ or $\mathtt{UA}_2$ and $\mathtt{OA}_2$. In the current authorization graph of figure 4.1, all operations that creates assignment relations can grant *Bob*'s requested access only when combined with the association relation of the sets $\mathtt{UA}_2$ and $\mathtt{OA}_3$ or $\mathtt{UA}_2$ and $\mathtt{OA}_2$. We can combine one of the operations that create user assignment, user attribute, and object attribute assignments with the mentioned association. The pair of edges, user assignment, association from user to object attribute, user attribute assignment, association from user to object attribute, and association from user to object attribute, object attribute assignment are relations that can authorize a *source user* request using two operations.

Further, a three operations approach that create a triple of edges, user assignment, user attribute assignment, association from user to object attribute, user assignment, association from user to object attribute, object attribute assignment, and user attribute assignment, association from user attribute to object attribute, object attribute assignment can authorize a request on *target policy element* using the sets of edges {(*source user*, $\mathtt{UA}_3$), ($\mathtt{UA}_2$, $\mathtt{UA}_3$), ($\mathtt{UA}_3$, $\mathtt{OA}_3$)}, {(*source user*, UA$_3$),(UA$_3$, OA$_1$), (OA$_2$, OA$_1$)}, {(*source user*, UA$_3$), (UA$_2$, UA$_3$), (UA$_3$, OA$_1$), (OA$_2$, OA$_1$)}, respectively.

4.4 Algorithmic Design and Evaluation

4.4.1 Policy Review Algorithms for Constrained Authorization and Revocation

This algorithm starts with the *main* function for constrained authorization and revocation of access query. It takes as input a request, areq$_d$ = (source, op$_x$, target), where the source is the user requesting access, op$_x$ is the operation the user want to perform on the (protected resource) target. An authorization graph G$_{auth}$ = (PE, ASSIGN, ASSOCIATION) associated with the request and the key-value record authMode for specifying constraints are the two other inputs. The output of the algorithm is the sets of operations that can grant source request if not authorized yet. Otherwise, the output is the sets of operations that can revoke source request has been authorized.

The first function call from the *main* is the isRequestAuthorized that returns result a boolean value of true if the *source user* request has not been authorized and the authAssoc, a set of associations that can grant the source request on target. If result is true and the target is a policy element of user or attribute type, the *getUAOASetsHelper* is called. The *getUASetsHelper* generates the access enabling user attribute sets. The elements of the access enabling user attribute sets assigned in the *getUASetsHelper* function according to the key-value pairs in the authMode input parameter.

These user attribute sets, the source, the output set $AREQ_A$, and the subsets of operation that can authorize source request are input parameters to the function *targetContainsSource*. In this condition that the value of result is true (see Step 2.1a), the *targetContainsSource* returns operations that creates relations to authorize source access on $AREQ_A$.

If the value of result is true but the target is an object or object attribute, the algorithm executes the block of code in the Step 2.1b. OPS is set to the values to create assignments (i.e., user assignment, user and object assignment) and associations from user attributes to object attributes.

Algorithm 4.1 Main

```
Input: $G_{auth}$ = (PE, ASSIGN, ASSOCIATION),
    areq$_d$ = (source, op$_x$, target),
    authMode = $\{$isGeneric: true/false, denySet: UA$_2$/UA$_3$, limitAccess: Successors / Path$\}$
Output: $AREQ_A$
Step 1: /*Initialize variables*/
    $AREQ_A \leftarrow$ []
    OPS $\leftarrow$ { }
Step 2: /*Check if request is for authorization or revocation and assign key-value for operation record/
  1. result, authAssoc = isRequestAuthorized($G_{auth}$, areq$_d$
    a. If result and target $\in \{U \cup UA\}$
          OPS = {assign: createAssign, assoc: createAssoc}
          UA$_1$, UA$_2$, UA$_3$, UA$_4$ $\leftarrow$ getUASetsHelper($G_{auth}$request, authAssoc, authMode)
          Let OP$_{subsets}$ $\leftarrow$ {op : op $\in 2^{op_j} \wedge$ op$_x$  op }
          Return targetContainsSource(source, $AREQ_A$, UA$_1$, UA$_2$, UA$_3$, UA$_4$, OP$_{subsets}$)
    b. Else if result and target $\in \{O \cup OA\}$
          OPS = {assign: createAssign, assoc: createAssoc}
          UA$_1$, UA$_2$, UA$_3$, OA$_1$, OA$_2$, OA$_3$ $\leftarrow$ getUAOASetsHelper($G_{auth}$request, authAssoc, authMode)
          Let OP$_{subsets}$ $\leftarrow$ {op : op $\in 2^{op_j} \wedge$ op$_x$  op }
          Return sourceNotContainedByTarget(source, $AREQ_A$, UA$_1$, UA$_2$, UA$_3$, OA$_1$, OA$_2$, OA$_3$, OP$_{subsets}$)
    c. Else
          OPS = {assign: deleteAssign, assoc: deleteAssoc}
          Return getRevocationApproaches($G_{auth}$, areq$_d$, ASSOCIATION)
```

The *getUAOASetsHelper* utilizes the values in authMode to determine if the query is for authorization with or without constraints, and assign values to the access enabling attribute sets. The last function call in this block of code, *sourceNotContainedByTarget*, returns sets of operations that create relations to allow source perform op$_x$ on target.

Algorithm 4.2 getUASetsHelper

```
Input: areq_d, authAssoc, authMode
Output: UA_1, UA_2, UA_3, UA_4

1. If authMode[isGeneric]
       CanGenerate = {}
       Return getUASets(areq_d, authAssoc, authMode, CanGenerate)
2. Else If Not authMode[isGeneric] ∧ authMode[denySet] == UA_3
       CanGenerate = { UA_2: true, UA_3: false }
       Return getUASets(areq_d, authAssoc, authMode[limitAccess], CanGenerate)
3. Else If Not authMode[isGeneric] ∧ authMode[denySet] == UA_2
       CanGenerate = { UA_2: false, UA_3: true }
       Return getUASets(areq_d, authAssoc, authMode[limitAccess], CanGenerate)
```

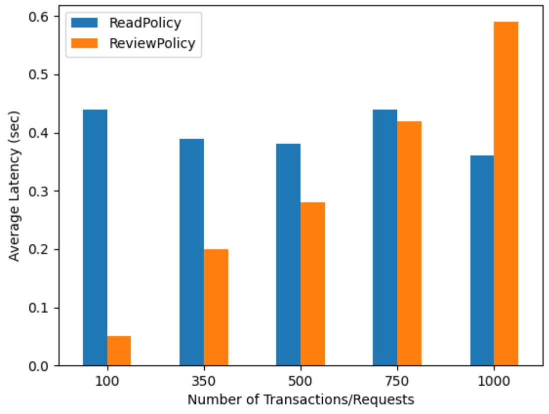


Figure 4.2: Average latency for number transactions and access requests

In a situation that the isRequestAuthorized returns a value of false for `result`, the *getRevocationApproaches* is called after setting the values of `OPS` to delete operations. The *getRevocationApproaches* has three subroutines, *canRevokeByUserAttribute*(), *canRevokeByAttribute*(), and *canRevokeByAssociation*(). These subroutines query for revocation approaches on any authorized request.

Algorithm 4.3 getUASets

```
Input: request, authAssoc, authMode, CanGenerate

Output: $UA_1, UA_2, UA_3, UA_4$

$UA_1 = \{ua : ua == authAssoc[pre] \vee ua \in \textit{ancestors}(authAssoc[pre])\}$

$UA_2 = \{ua : ua \in \textit{descendants}(request[source])\}$

$UA_3 = \{ua : ua \in \textit{ancestors}(authAssoc[suc]), ua \notin \textit{ancestorsOfSet}(UA_2), ua \notin UA_2, ua \notin \textit{descendantsOfSet}(UA_1), ua \notin UA_1\}$

$UA_4 = \{ua : ua \in \textit{descendants}(request[source]) \wedge ua \in \textit{descendants}(request[target])\}$

  1. If authMode[isGeneric]
         Return $UA_1, UA_2, UA_3, UA_4$
  2. Else If Not authMode[isGeneric]
      a. If $CanGenerate[UA_2] \wedge limitAccess[key] == Successors$
            $UA_2 = \{ua : ua \in UA_2 \wedge ua \in limitAccess[value]\}$
      b. Else If $CanGenerate[UA_2] \wedge limitAccess[key] == Path$
            $UA_2 = \{ua : ua \in UA_2 \wedge ua \in \textit{descendants}(limitAccess[value][pathSource]) \wedge ua \in \textit{ancestors}(limitAccess[value][pathSink])\}$
      c. Else
            $UA_2 = \{\}$
      d. If $CanGenerate[UA_3] \wedge limitAccess[key] == Successors$
            $UA_3 = \{ua : ua \in UA_3 \wedge ua \in limitAccess[value]\}$
      e. Else If $CanGenerate[UA_3]$ $limitAccess[key] == Path$
            $UA_3 = \{ua : ua \in UA_3 \wedge ua \in \textit{descendants}(limitAccess[value][pathSource]) \wedge ua \in \textit{ancestors}(limitAccess[value][pathSink])\}$
      f. Else
            $UA_3 = \{\}$
      g. Return $UA_1, UA_2, UA_3, UA_4$
```

Algorithm 4.4 targetContainsSource

```
Input: source, authorizationApproaches, $UA_1$, $UA_2$, $UA_3$, $UA_4$, $OP_{subsets}$, $op_x$

Output: authorizationApproaches

/*Check for possible authorization approaches*/

  1. If {OPS[assign]} $\subseteq$ $OP_{subsets}$
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, source, $UA_1$}
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, $UA_2$, $UA_1$}
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, source, $UA_3$}, {PU, {OPS[assign]}, $UA_3$, $UA_1$}
  2. If {OPS[assoc]} $\subseteq$ $OP_{subsets}$
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assoc]}, $UA_2$, $op_x$, $UA_4$}
  3. If {OPS[assign], OPS[assoc]} $\subseteq$ $OP_{subsets}$
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, source, $UA_3$}, {PU, {OPS[assoc]}, $UA_3$, $op_x$, $UA_4$}
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, $UA_2$, $UA_3$}, {PU, {OPS[assoc]}, $UA_3$, $op_x$, $UA_4$}
        authorizationApproaches = authorizationApproaches $\cup$ {PU, {OPS[assign]}, source, $UA_3$}, {PU, {OPS[assign]}, $UA_1$, $UA_3$}, {PU, {OPS[assoc]}, $UA_3$, $op_x$, $UA_4$}
  4. Return authorizationApproaches
```

Algorithm 4.5 canRevokeByUserAttribute

```
revocationApproaches $\leftarrow$ { }

sourceNeighbors $\leftarrow$ {}

  1. If outDegree(source) > 1
       a. sourceNeighbors = sourceNeighbors $\cup$ successors(source)
       b. For ua $\in$ sourceNeighbors
            If ua $\in$ anc(target)
              revocationApproaches = revocationApproaches $\cup$ (deleteAssign, source, ua)
       c. Return revocationApproaches
```

Algorithm 4.6 canRevokeByAttribute

```
MIN = 1
revocationApproaches ← { }
uaNeighbors ← { }
sourceToTargetPath ← $des$(source user) ∩ $anc$(target)
  1. For ua ∈ sourceToTargetPath
         If outDegree(ua) > MIN; uaNeighbors[ua] = successors(ua)
  2. For key, value, ∈ uaNeighbors
         For node ∈ value
             If node ∈ $anc$(target) ∧ inDegree(node) > MIN
                 revocationApproaches = revocationApproaches ∪ (deleteAssign, key, node)
  3. Return revocationApproaches
```

4.4.2 Performance Evaluation of Constrained Access Authorization and Revocation in Hyperledger Fabric

In this section, we present the details of our experiments carried out for performance evaluation. The experiments were in two steps, an on-chain that reads the Policy Information Ledger and an off-chain policy review analysis. An iterative process in the policy review algorithm will degrade the blockchain network performance if deployed to the network. We performed this experiment using a virtual machine with 2 CPUs, 10GB RAM, running Ubuntu 16.04 LTS operation system, and Hyperledger Fabric V2.2 installed. We built a Fabric blockchain testbed with one Raft orderer service node, two peers for an organization on a single channel, and a LevelDB database.

We created a policy graph generator script that simulates the creation of policy elements to the Policy Information Ledger. The policy graph comprises a policy class, 300 user and object attributes, and 200 users and objects. We generated workloads for the read policy graph transaction into the Fabric blockchain using the Hyperledger Caliper V0.4.2. Hyperledger Caliper is a

Algorithm 4.7 getRevocationApproaches

```
Input: G_auth, areq_d, ASSOCIATION
Output: revocationApproaches
Step 1: /*Initialize the revocation approaches set and make function calls*/
revocationApproaches ← { }
canDeleteUserAssignment ← canRevokeByUserAttribute(G_auth, areq_d)
canDeleteAttributeAssignment ← canRevokeByAttribute(G_auth, areq_d)
canDeleteAssociation ← canRevokeByAssociation(G_auth, areq_d, ASSOCIATION)
Step 2: /*combine approaches if possible*/
  1. If canDeleteUserAssignment; revocationApproaches = revocationApproaches ∪ canDeleteUserAssignment
  2. If canDeleteAttributeAssignment; revocationApproaches = revocationApproaches ∪ canDeleteAttributeAssignment;
  3. If canDeleteAssociation; revocationApproaches = revocationApproaches ∪ canDeleteAssociation
  4. If canDeleteUserAssignment ∧ canDeleteAttributeAssignment
     revocationApproaches = revocationApproaches ∪ canDeleteUserAssignment · canDeleteAttributeAssignment
  5. If canDeleteUserAssignment ∧ canDeleteAssociation
     revocationApproaches = revocationApproaches ∪ canDeleteUserAssignment · canDeleteAssociation
  6. If canDeleteAttributeAssignment ∧ canDeleteAssociation
     revocationApproaches = revocationApproaches ∪ canDeleteAttributeAssignment · canDeleteAssociation
  7. If canDeleteUserAssignment ∧ canDeleteAttributeAssignment ∧ canDeleteAssociation
     revocationApproaches = revocationApproaches ∪ canDeleteUserAssignment · canDeleteAttributeAssignment · canDeleteAssociation
  8. Return revocationApproaches
```

blockchain performance benchmark framework, which allows users to test different blockchain solutions with customized use cases and get a set of performance test results. To test the performance of our algorithm another script reads the policy graph ledger, simulates requests for authorization and revocation, and sets values for authorization mode record. The graph in Figure 4.2 shows the average latency for reading the policy graph using the Caliper. Also, on the same graph, the average response time to generate revocation and constrained authorization approaches for the request sizes are shown. The *policyRead* average latency varies in the range of 0.36 to 0.44 seconds for the number of transactions. The average response time of the *policyReview* increases as the number of requests for revocation and constrain authorization increases.

CHAPTER 5: IMPLEMENTATION OF ABAC REVIEW IN HYPERLEDGER FABRIC BLOCKCHAIN

Blockchain was invented in 2008 for bitcoin, the first digital currency to solve the double-spending problem without the need for a trusted authority or central server [42]. The invention of bitcoin (BCH) inspires the likes of Ethereum(ETH) [9], Litecoin (LTC) [65], Tether (USDT) [38], and much more as cryptocurrencies. In less than a decade, the blockchain distributed ledger technology has evolved and paved the way for other usages, apart from cryptocurrency. Other non-crypto currency blockchain platforms are Corda [7] and Hyperledger Fabric [10].

In this chapter, we will implement an Attribute-Based Access control ABAC review in Hyperledger Fabric HLF. Hyperledger Fabric (HLF) is an open-source distributed ledger software that provides the framework for developing applications or solutions with a modular architecture. In the sections that follow, we briefly discuss the general underlying concepts of blockchain as a distributed ledger and describe the integral components of the Hyperledger Fabric. We provide the process of developing the HLF components into a network for application development and describe the typical flow of business transactions in the HLF network. We also touch on the default access control in the HLF and motivate the need for an ABAC approach.

We intend to implement the policy machine, an instance of the ABAC model, in the Hyperledger Fabric. This effort will include integrating the authorization and revocation review algorithms from the previous chapters into the HLF network. To conclude this chapter, we will evaluate the performance of the ABAC review implementation in the Hyperledger Fabric.

5.1 Background

Blockchain can be defined as an immutable decentralized ledger for recording transactions, maintained within a distributed network of mutually untrusted peers. Every peer execute a consensus protocol to validate transactions grouped into blocks and maintains a copy of the ledger that constitute the blocks. Blockchain is temper-proof because every new block is cryptographically linked to the previous one.

Blockchain is an emerging technology that can radically improve secured transparent transactions at banking, supply chain, and other transaction networks. Itâs estimated that Blockchain will generate \$3.1 trillion in new business value by 2030[1]. The Distributed Ledger Technology (DLT) is the basis for the application of blockchain to save time when recording transactions between parties, remove costs associated with intermediaries, and reduce risks of fraud and tampering. Some of the well known Blockchain platforms includes hyperledger fabric, ethereum, corda, etc. Generally, blockchains are classified as permissionless and permissioned blockchain. Permissionless blockchains such as Bitcoin and ethereum allow anonymous node to join the blockchain network without permission. On the other hand, hyperledger fabric and corda are examples of permissioned blockchain that require identifiable nodes to obtain permission from a federated authority(ies) before joining the blockchain network.

Generally, there are four building blocks of a blockchain framework that includes a shared ledger, cryptography, system of trust or consensus, and business rules or smart contracts. The blockchain frameworks utilize a configured SQL or noSQL distributed database that conforms with its “append only” or immutable protocol. Authentication, traceability, and verifiability of the business transactions on a blockchain network are achieved using cryptography. The consensus protocol provides a system for all participating (blockchain network entities) peers to agree on a value as an outcome of a business transaction. Smart contract are executable business terms that are embedded in a blockchain transaction database and defines the flow of value and state of each transaction.

5.2 Hyperledger Fabric Overview

We provide an overview of components the Hyperledger Fabric network design is comprised of in this section. The core infrastructural components are the peer nodes and orderer nodes. Other components are the client node, chaincode (smart contract) ledger, and channels. These components are briefly described.

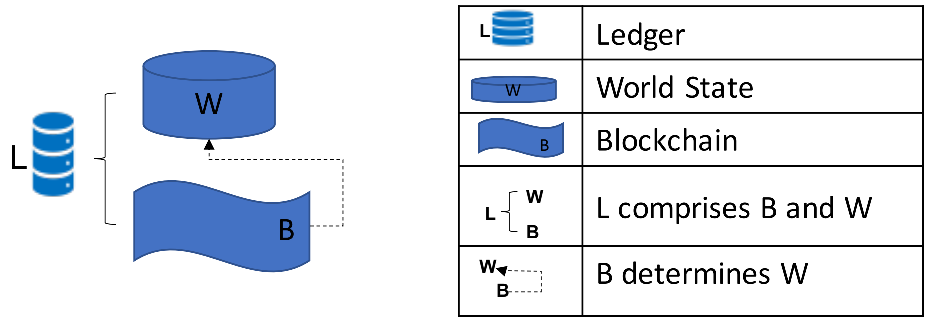


Figure 5.1: A Ledger L comprises Blockchain B and world state W

Ledger

Hyperledger Fabric ledger stores assets. An asset (or business object) is the representation for anything of monetary value stored on a blockchain network and is transferable between blockchain network participants. A ledger comprised of the world state (or simply state) and the blockchain, see figure 5.1. The former is a database expressed as key-value pairs that holds the current value of an asset. A key is a unique name for an asset. The value are attributes that describe the asset. The world state provides an easy and direct access to the current value of an asset in a ledger rather than having to traverse the assetâs history in the blockchain. The blockchain is structured as sequential log of interlinked blocks, where each block contains a sequence of transactions, each transaction representing an update to the world state.

Nodes

There are three types of nodes in Hyperledger Fabric network. A client (client node) acts on behalf of the end user for creating and submitting request to the network. Clients are created using Hyperledger Fabric SDKs. Peer nodes (or simply peers) are the fundamental element of the Hyperledger Fabric blockchain network, they host the ledgers that store the asset. Nodes that render the service of block sequencing, as well as transaction sequence within blocks when blocks are first created in Hyperledger Fabric blockchain network are called ordering node. An ordering node is also used to initialize network. At the launch of an ordering node, it produces the first block written to the ledger, genesis block. The genesis block contains the network configuration properties and policies specified at the initialization of the orderer node.

Chaincode

A chaincode is an implementation of smart contract in Hyperledger Fabric, a piece of code that accesses the ledger and provides instructions for the asset and network component modification. A transaction is an action that triggers the change of an assetâs value or the blockchain network configuration. While business transaction (or simply transaction) alters an asset value, configuration transaction enables updates to the blockchain network components. similarly, a chaincode that specify the logic for asset queries and updates are called application chaincode. These are installed and instantiated as isolated processes on peer nodes. Whereas, system chaincode are smart contract that specify logic for modifying components of blockchain network. System chaincodes are programmed into the peer node binary at inspection.

Channels

An instance of the blockchain network is called a channel, it serves as a mechanism for a set of components to communicate and transact privately. There are two types of channel. he first channel that is created in a Fabric network is the “system” channel. The system channel defines the set of ordering nodes that form the ordering service and the set of organizations that serve as ordering service administrators. Peers transact on private “application” channels that are derived from the ordering service system channel.

Membership Service Provider (MSP)

The Membership Service Provider (MSP) refers to an abstract component of the system that provides credentials to clients, and peers for them to participate in a Hyperledger Fabric network. Clients use these credentials to authenticate their transactions, and peers use these credentials to authenticate transaction processing results (endorsements). While strongly connected to the transaction processing components of the systems, this interface aims to have membership services

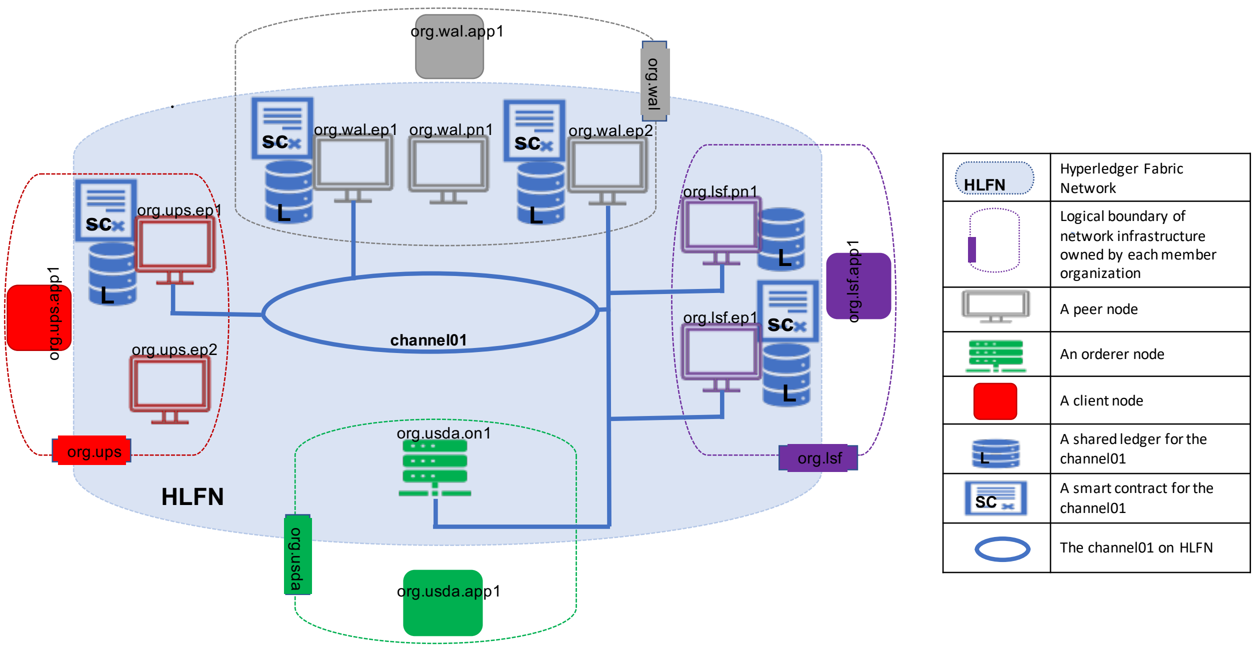


Figure 5.2: Network infrastructure partitioned by member organizations and their connection to a channel

components defined, in such a way that alternate implementations of this can be smoothly plugged in without modifying the core of transaction processing components of the system [1].

Organization

Also known as "members", organizations are invited to join the blockchain network by a blockchain network provider. An organization is joined to a network by adding its Membership Service Provider (MSP) to the network. The MSP defines how other members of the network may verify that signatures (such as those over transactions) were generated by a valid identity, issued by that organization [1]. The particular access rights of identities within an MSP are governed by policies which are also agreed upon when the organization is joined to the network.

Figure 5.2 is a minimal representation of the HLF supply chain network. The network consists of four organizations, United State Department of Agriculture (USDA, org.usda) as the regulator, Lone Star Farms (org.lsf) as the producer, Walmart (org.wal) as the retailer, and UPS (org.ups) as the shipper. The ordering node, org.usda.on1 provides the ordering service for the HLF network.

The peers, org.lsf.ep1, org.wal.ep1, org.wal.ep2, and org.ups.ep1 are nodes that can invoke update transactions because they have the chaincode (smart contract) for channel01 installed. Peer org.lsf.pn1 connects to channe01, can respond to a query transaction proposal and not update transaction proposal it has the ledger L for channel01. Multiple channels can exist on a network. Nodes without a link to channel01 are participating in another channel not shown in figure 5.2. A peer node can serve multiple channels, as they are isolated from each other.

5.3 Hyperledger Fabric Blockchain Network Development

Severs that host infrastructural components of Hyperledger Fabric are required to have (os-virtualization software) docker container installed. At runtime, the creation of chaincode (smart contract) is launched in isolated Docker containers. Secondly, HLF infrastructure components (orderer and peers) are developed as docker images. Another reason is that docker container orchestration system, Kubernetes is used to setup the Hyperledger Fabric network. We now provide an overview of HLF executable components and tools that exist as Docker images. These executable components and tools require configuration information provided in the YAML file format.

5.3.1 Tools and Components

The development of HLF network relies on tools such as the configtxgen, configxlator, and cryptogen. Configtxgen is a utility tool for managing configuration artifacts such as genesis block (orderer channel block) and transaction channel. Cryptogen tool generates cypto material, digital certificate and key-store, in a testing environment for users, network infrastructure, and member organization. Configxlator serves as a utility tool to translate configuration encoded in protocol buffer to a readable (JSON) format.

The peer binary is a process that serves as a node in the HLF network. It is also used as a tool to manage network and channel level configurations. A peer node maintains the ledge and manages the chaincode (smart contract) delayed to it. Peer node exposes services build on grpc and the services are invoked by the clients, other peers, and the orderer for sending blocks to the peer. Peer nodes exchanged data by utilizing a gossip dissemination protocol. The orderer binary is single file that requires the genesis block for it initialization. The runtime properties are controlled in a (YAML) configuration file. The HLF framework provides a messaging system in the ordererâs node for the ordering and distribution of blocks using Raft. The type and specification of an orderer nodes is controlled by the configuration file, orderer.yaml. The orderer binary also exposes services to peers and client through the grpc and depends on the use of crypto service provider for encryption, decryption, and signing of messages.

5.3.2 Network Initialization

HLF network is only launched through an orderer node. A designated server for the ordering service is initialized with crypto-config.yaml, configtx.yaml, and orderer.yaml configuration files. The configuration transaction file, configtx.yaml specifies which organizations are members of a channel (defined as a consortium), the ordering nodes that can add new blocks on a given channel, as well as the policies that govern channel updates. The initial channel configuration is stored in the ledger as the first block, also referred to as channel genesis block, and the genesis block is updated through channel configuration updates. After a successful launch of the orderer node, peer nodes of organizations defined in the consortium or consortia can join a predefined channel or request the creation of a new channel. To facilitate the storage of an asset into channel ledger and usersâ business transaction on an asset, the consortium associated with the channel specifies the development of a chaincode (smart contract) that is installed on a channel peer node. In addition, consortiumâs application developer creates application installed on client nodes using the HLF SDKs that enable end user to create transaction request and submit to the network.

5.3.3 Transaction flow

After the installation of chaincode dedicated for a business network on peer node and conforming application is available on the client node, end users can invoke the chaincode (smart contract) to read and update the ledger. A ledger query is a straight forward transaction that a response is returned immediately. A peer can return the results of a query to a client application instantly since

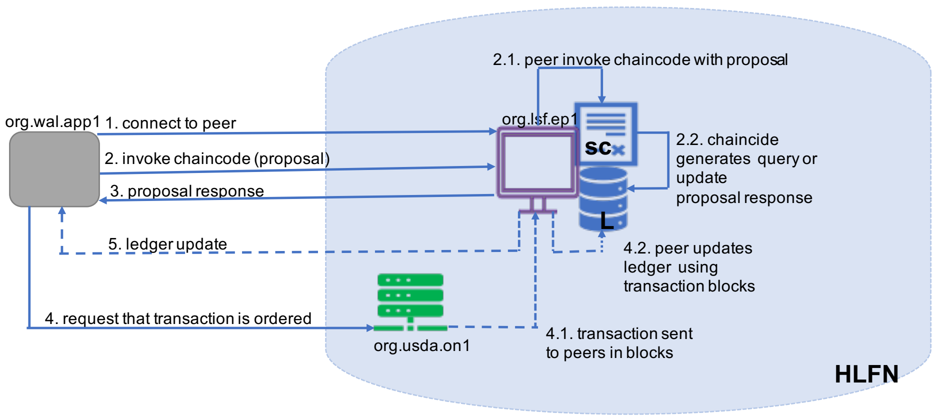

2.1. peer invoke chaincode with proposal
org.wal.app1
1. connect to peer
org.lsf.ep1
SC
2.2. chaincode generates query or update proposal response
2. invoke chaincode (proposal)
3. proposal response
L
5. ledger update
4.2. peer updates ledger using transaction blocks
4. request that transaction is ordered
org.usda.on1
4.1. transaction sent to peers in blocks
HLFN

Figure 5.3: The transaction flow for query and update proposal to the ledger

the peerâs local copy of the ledger has the required information to satisfy the query. Figure 5.3 shows a query request in three steps - the client application connects to a peer, invokes a chaincode with a query request, and the chaincode returns a query response. However, ledger update is a more complex interaction between the client application, peers and orderer. In addition to three steps in the query transaction request, two extra steps are required in an update transaction request. A single peer node can not respond to a ledger update transaction request, it is a consensus process among all the peers that has the ledgeâs copy. The designated chaincode on peer node returns a simulated (i.e., results not effected on the ledger) results as step three of a ledger update transaction request.

5.4 Access control in HLF

HLF uses access control lists (ACLs) to manage access to resources by associating a policy with a resource. The resources protected through ACLs are stream of events, functions exposed by both application and system chaincodes. Users access these resources using the client application or the peer tools and policies are used to exercise access control on the resources (endpoints). In the appropriate section of the configuration file, configtx.yaml, a resource represented in the convention <component>/<resource> is mapped to a policy. Where <component> is an HLF module and the <resource> is a function exposed by the module. For example, peer/Propose represents a target resource âProposeâ for invoking a transaction and it is a function exposed through the âpeerâ module

5.4.1 Hierarchies, Naming, and Types of Policy

Policies are specified at different sections of the configtx.yaml file. These policies are encoded in the genesis block of the HLF network. The encoded policies translate to the hierarchy of policies specified at different levels of the network. HLF fabric adopt the unix-like representation of file path to describe the hierarchy of policies. Hierarchy of policies are in three tiers, the top most is /Channel, second tier policies are /Channel/Application and /Channel/Orderer, and the last tiers are /Channel/Application/<org id>/ and /Channel/Orderer/<org id>. Where <org id> is the unique name (identity) of an organization in the channel. For instance, a policy specified at the /Channel level defines rules that governs access to channel level resources.

Policies have names. The fully qualified name of a policy is determined by the hierarchy followed by policy name (i.e., policy name is the leaf level of hierarchy). The standard policy names are Readers, Writers, Admins, and Endorsers. For example, /Channel/Readers defines a Readers policy with a rule that applies to a system channel level resource.

A policy definition is comprised of the policy type and the rule. The policy types are signature and implicit meta policies. For the signature policy types, the rules are boolean expression in terms of principle, using boolean functions OR(..), AND(..), and OUTOF(..). Signature policies directly evaluates to a true (grant) or false (deny). On the other hand, implicit meta policies are used for creating consensus driven decision points. Instead of the boolean expression, the rule of an implicit meta policy type refers to sub policies (i.e., refers to another implicit meta or a signature policy), using the keyword ANY, ALL, and MAJORITY. The evaluation of implicit meta policy at runtime is an aggregated results from referred sub policies.

5.4.2 Motivation for ABAC in Hyperledger Fabric Blockchain Network

Using the default Hyperledger Fabric access control, it is cumbersome for administrators to specify flexible access control based on asset (business object) attributes. For every use case, application

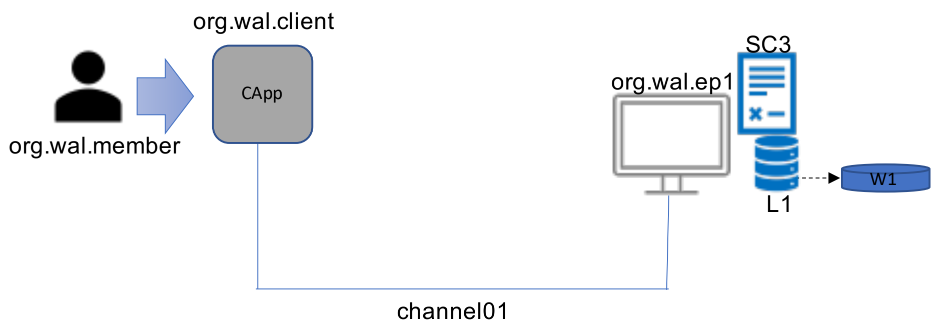
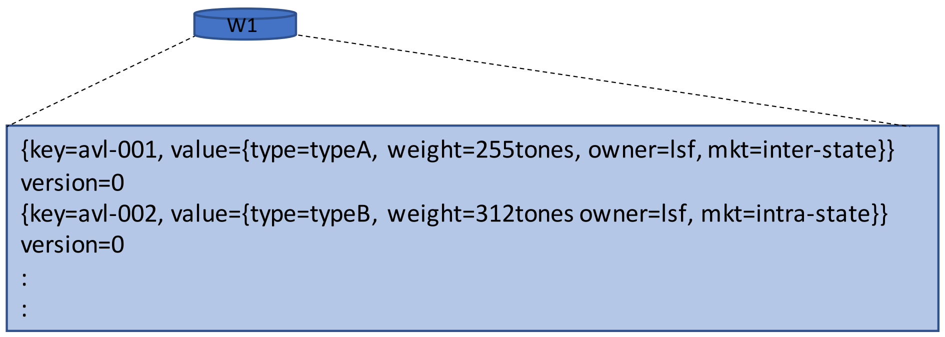


Figure 5.4: Partial Components of a Channel, channel01



```
{key=avl-001, value={type=typeA, weight=255tones, owner=lsf, mkt=inter-state}}
version=0
{key=avl-002, value={type=typeB, weight=312tones owner=lsf, mkt=intra-state}}
version=0
:
:
```

Figure 5.5: World state W1 for Ledger L1 contains two business objects

chaincode (smart contract) developers are tasked with the development of access control policy for complex business objects. We demonstrated this claim in the scenario that follows.

5.4.3 Coarse Access control Scenario in HLF

From figure 5.4, using the default ACLs, assuming the principal org.wal.member has the authority to propose a transaction on the ledger L1 by invoking the smart contract SC3 that changes the value of an asset in the state W1.

It is cumbersome to use ACL if it is required to restrict the access of the principal org.wal.member to asset values of the attribute typeB (see figure). Any selective access to HLF resource is dealt with on a case by case basis by an application chaincode developer.

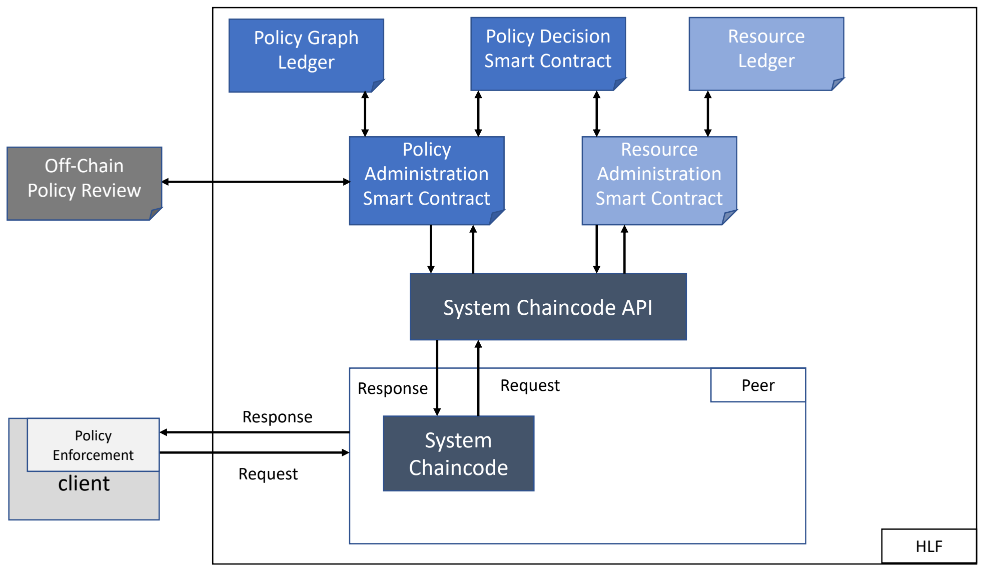


Figure 5.6: Blockchain Access Control System Architecture.

5.5 System Architecture and Implementation

In section 2.5, we provide an overview of the Policy Machine. Figure 5.6 represents the functional architectural implementation of Policy Machine in the Hyperledger Fabric blockchain network. We implement the Policy Information Point (PIP) for storing the access control state and the database for the protected resources as Policy Information and Resource Ledgers, respectively. The Policy Administration Smart Contract represents the Policy Administration Point (PAP). It mediates access and enables modification to the Policy Information Ledger. Similarly, we implement the (RAP) as Resource Access Smart Contract that intercepts application (resource) user's request to the Resource Ledge. Our Policy Decision Smart Contract (see figure 5.6) is the logic that represents the Policy Decision Point (PDP) component of the Policy Machine. It makes access decisions on access requests forwarded from both Policy Administration Smart Contract and Resource Access Smart Contract response.

This architecture aims at implementing the Policy Machine for client nodes that are native to the blockchain. We need not implement a smart contract for the Policy Enforcement Point (PEP), and the Event Processing Point (EPP) component of the Policy Machine is outside the scope of this work.

Some components of the Policy Machine we implement as Smart contracts are in the go programming language. We leverage the invokeChaincode Application Programming Interface (API) to enable the request and response between Smart contracts (chaincode). For instance, assuming the installation of the smart contracts are on the same channel. The Resource Access Smart Contract conveys the access decision for an application users' request. It locally calls the invoke function of the Policy Decision Smart contract that returns access decision response. This inter-chaincode function call does not require a new transaction message. The calling chaincode uses the same transaction context as its caller. Note that invokeChaincode API allows the cases where the calling and called chaincode are on the same or different channel. However, for our implementation, only when an invocation of a chaincode by another on the same channel is allowed. Recall that the Policy Information Ledger stores an abstract representation of protected resources in the Resource Ledger. For consistency on these two ledgers, the Policy Decision Smart contract is allowed both read and write access on the two ledges. If the Policy Decision Smart Contract is not on the same channel with the two ledges, any (delete/create) modification request to these ledges will not affect.

5.6 Use Case And Scenarios

A case study for this work is a blockchain-based global Real Estate Investment Trust (REIT). The concept of global REIT is a portfolio diversifier designed to deliver high returns and income through investments in real estate investment trust (REIT) and real estate companies worldwide. It offers investors exposure to global real estate markets without the necessity of acquiring an entire property and shift the management and compliance obligations to the fund management. Apart from offering high total investment return through a combination of capital appreciation and current income like traditional REITS, blockchain-based global Real Estate Investment Trust (e.g., globalreit.co) also provide (a) real estate investment using cryptocurrency (b) stable dividends for

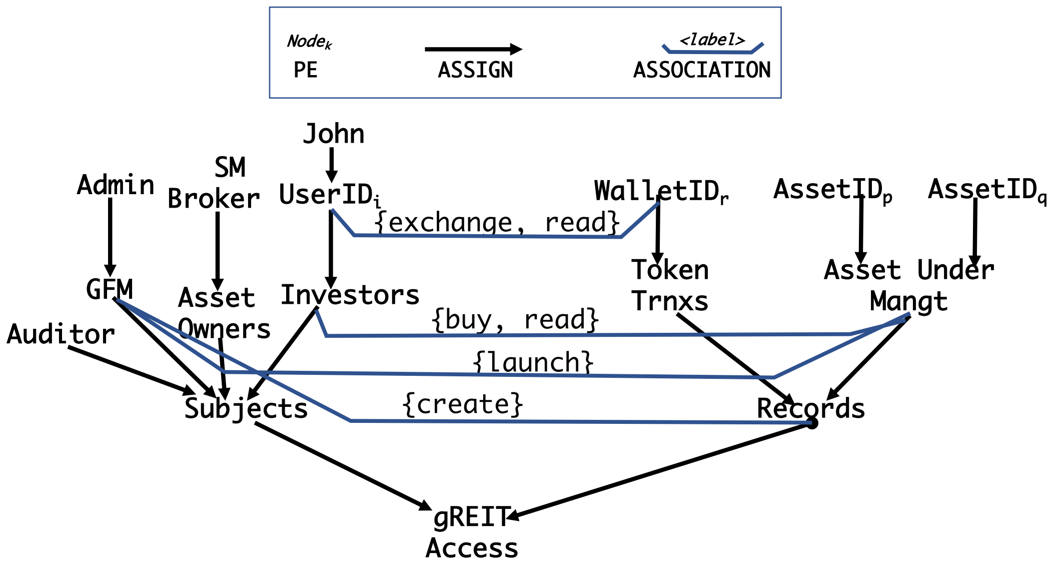


Figure 5.7: Policy Machine Authorization Graph for global REIT

Scenario 1

```
[redacted]$ source John-login.sh
[redacted]$ peer chaincode query -C pmchannel -n
ttcc  -c '{"Args":["readAccount", "WalletID"]}'
{"ID":"WalletID","GREIT":"500","ERC20":"2000","accountType":"invest"}
[redacted]$ peer chaincode invoke
"${TARGET_TLS_OPTIONS[@]}" -C pmchannel -n ttcc  -c '{"Args":["buyGRET",
"WalletID", "KJKPlaza", "150"]}'
2021-05-08 13:13:24.756 CDT [chaincodeCmd] chaincodeInvokeOrQuery -> INFO 001
Chaincode invoke successful. result: status:200 payload:"650"
```

Scenario 2

```
[redacted]$ source SMBroker-login.sh
[redacted]$ peer chaincode invoke
"${TARGET_TLS_OPTIONS[@]}" -C pmchannel -n pgacc  -c
'{"Args":["launchTokenizedAsset", "UpTownHotel", "AUM", "gREITAccess"]}'
Error: endorsement failure during invoke. response: status:500 message:"SMBroker
is unauthorized to launch token asset: UpTownHotel"
```

Figure 5.8: Policy Decision Smart contract response to access request scenarios

crypto investors (c) crypto domain for real estate asset holders to exit into a liquid market

A global REIT acquires assets around the world. Acquired assets, also called Asset Under

Management (AUM), with a Net Asset Value (NAV) and a projected portfolio value by the end of a specified time. The global REIT issues an Initial Coin Offering (ICO), the cryptocurrency industry equivalent of an Initial Public Offering (IPO). Investors interested in offering buy and receive a new cryptocurrency token issued by the company. This token may have some utility in using the product or service the company is offering, or it may just represent a stake in the company or project.

Our use case, a global REIT platform, utilizes three modules deployed to the Hyperledger Fabric network. The asset management module consists of an instance of the Resource Access Smart Contract called asset management chaincode and a ledger. It allows asset owners to register their assets with the Fund Manager. The fund manager completes the asset owner's background check and KYC compliance evaluation before the list (launch) of an asset on the platform. Investors have access to assets' information and choose to invest in assets listed on the global REIT platform.

The second module is the transaction module. It deploys a Resource Access Smart Contract called token transaction chaincode. The role of the transaction module is to receive and validates all investors' transaction requests to the platform. The access control module has a Policy Information Ledger and policy graph access chaincode. Policy Administration and the Policy Decision Smart Contracts constitute the policy graph chaincode. The following subsection explains the graphical representation of the Policy Information Ledger shown in Figure 5.7

5.6.1 Policy Machine Implemented for gREIT

Figure 5.7 represents a Policy Machine access control graph for the global REIT platform. An algebraic expression for the access graph is G = (PE, ASSIGN, ASSOCIATION). The graph nodes are elements of the set PE, policy element. Nodes on the left side of the graph represent participants (fund manager, asset owners, and investors) and their attributes. The nodes to the right represent the protected (object attributes) assets and transactions. While unlabeled edges (black arrows) of the access graph are elements of the ASSIGN, the labeled edges (blue arching lines) are the ASSOCIATION relations. Excluding the policy class node, gREIT, the assignment (unlabeled edges) from all other nodes must terminate at the gREIT Access node. The policy class is a container that holds all the access rules expressed in the access graph. An element of the ASSOCIATION relation is a triple (user attribute, access right set, object attribute). It implies that a user node with a sequence of assignment (unlabeled edges) that leads to the user attribute of an association can perform the operation(s) granted through access right set on the objects and object attributes that have a path (sequence of assignment) to the association object attribute. For instance, the association (GFM, create, Records) grants the user Admin the access right to perform the action that creates the real estate assets as tokens.

We have utilized the Hyperledger Fabric implementation of the ERC-20 (Etherum Request for Comments 20) for both the token value of an asset and investor's payment for an offering listing on the platform. The ERC-20 is a standard for Fungible Tokens. The records of tokenized assets reside in the ledger for the asset management module, while the record of the purchase of coin offering is the log to the ledger of the transaction module.

5.6.2 gREIT Access Request Scenarios

The access rights exchange and buy of the associations (UserIDi, exchange, read, WalletIDr) and (Investors, buy, read, Asset Under Mangt) in Figure 5.7 models the authorization of purchase by an investor. For an authorized investment in a token asset, an investor must pay from an account (WalletID) associated with the UserID of the investor. Scenario 1 of Figure 5.8 shows the interaction of the user John with the gREIT platform application. The query request 'readAccount' returns that John has 500 GREIT asset tokens and 2000 ERC20 currency tokens. His authorized 'buyGRET' transaction to invest 150 ERC20 currency in a tokenized asset 'KJKPlaza' response is an asset token balance of 650.

In another scenario, a user (SMBroker) requests to list (launch) an asset for the initial offering service. As the user SMBroker is not associated with the attribute (GFM) granted the access right to launch asset token, the Policy Decision Smart Contract denied the transaction request (see scenario 2 Figure in 5.8).

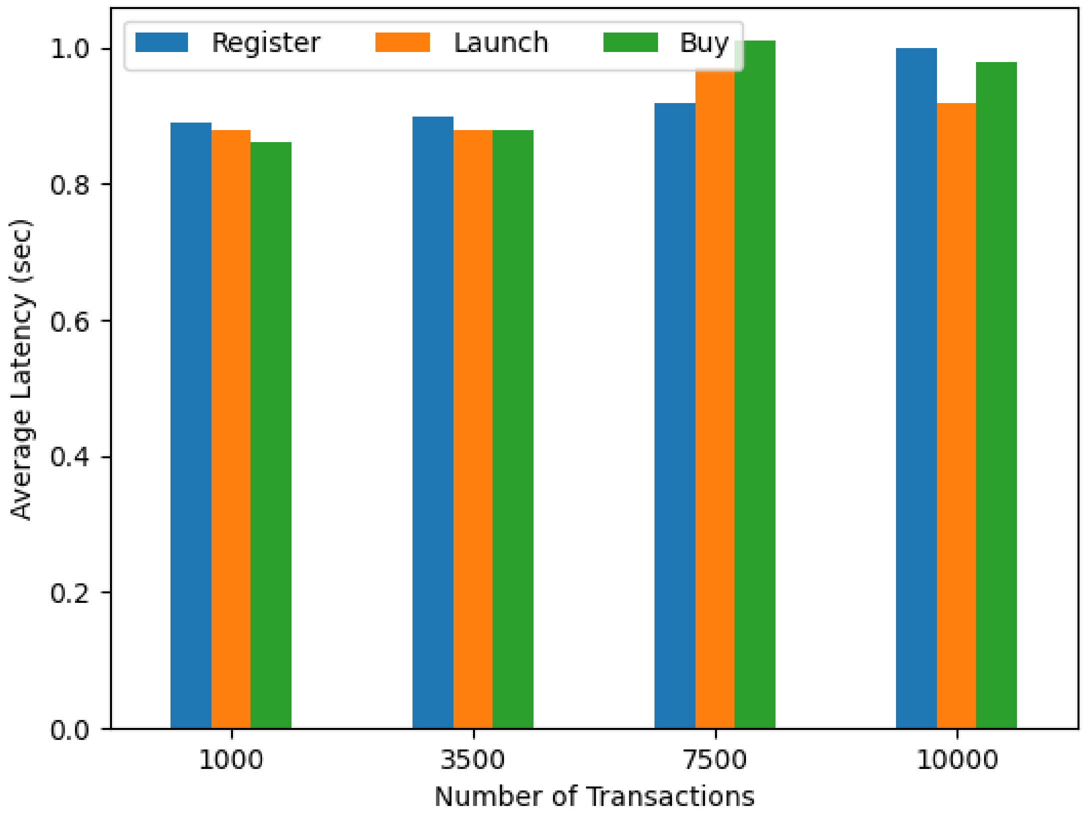


Figure 5.9: Average latency of register, launch, and buy transactions using LevelDB

5.7 Performance Evaluation of Policy Machine Implementation in Hyperledger Fabric

We performed this experiment using a virtual machine that has 2 CPUs, 10GB RAM, running Ubuntu 16.04 LTS operation system, and Hyperledger Fabric V2.2 installed. We built a Fabric blockchain testbed which has one Raft orderer service node, two peers for an organization on a single channel. The network configuration is evaluated against the two available data bases (LevelDB and CouchDB).

We generated the three types of transaction workloads into Fabric blockchain using the Hyperledger Caliper V0.4.2. Hyperledger caliper is a blockchain performance benchmark framework, which allows users to test different blockchain solutions with custom use cases, and get a set of

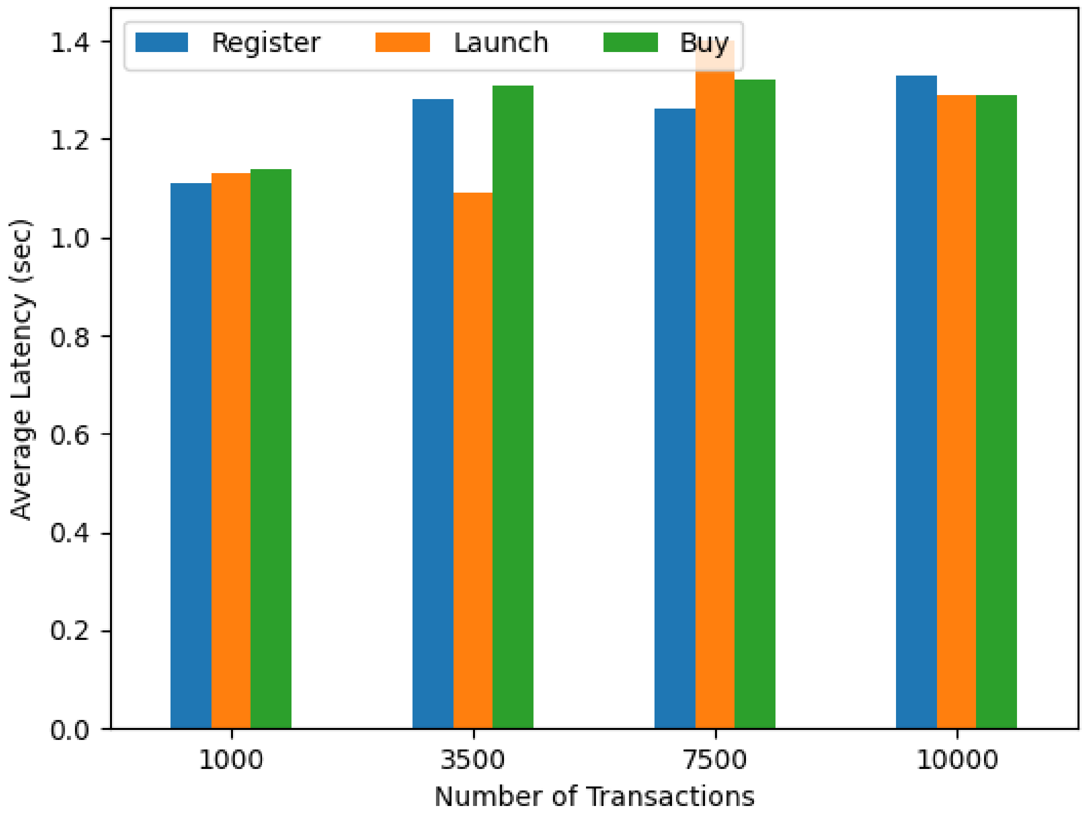


Figure 5.10: Average latency of register, launch, and buy transactions using CouchDB

performance test results. Caliper comprises two difference processes, a manager process and scalable worker process. The manager process initialize the Fabric network, schedules the configured rounds, and spools the performance report based on the observed transaction statistics. This experiment, our benchmark configuration sets the number of worker process to 20. For each round of execution ranging from 1000 to 10000 transactions evaluates the average latency for workloads of the transaction types (register, launch, buy).

Figures 5.9 and 5.10 plot the experimental results in terms of average transaction latency. The register transaction average latency increases linearly with an increase in transaction number. For 10000 transactions, the average latency for the LevelDB and CouchDB was 1.0 and 1.33 seconds, respectively. The two other types of transactions peaked when the transaction number was 7500. The average latency for the launch transaction was 0.97 and 1.40 seconds for the LevelDB and

CouchDB, respectively. Notice a similar trend for the buy transaction. CouchDB results in higher latency than the LevelDB since it incur internal network latency for the required HTTP communication. Conversely, the LevelDB can not effectively support complex schema and queries for the ledge.

CHAPTER 6: CONCLUSION & FUTURE WORK

The policy machine attribute-based access control model uses enumeration of attributes and relations to formulate policy. This feature makes performing (queries) reviews of policy feasible, rather than the theoretical NP-complete time complexity in models that express policy logically. However, the NIST reference specification is lacking policy reviews pertinent to the creation and modification of administrative access policy. An example in section 3.3 demonstrates how policy review using our proposed algorithm can prevent an unintended consequence in the PM administration. In section 3.4.2, the response time of the performed experiments indicates the proposed algorithm is scalable.

This work also proposed the NIST ABAC architecture and its implementation for a permissioned blockchain network. The policy Machine is well-adapted for a distributed system such as the blockchain, and it is scalable. After we discussed previous work that studies the implementation of ABAC on a blockchain, we presented the system architecture and implementation of the Policy Machine sub-components as smart contracts (chaincode). We provided a use case to demonstrate the feasibility of the Policy Machine for a blockchain network. The experimental evaluation of this work applies the two available types of databases for Hyperledger Fabric setup, and the Hyperledger Caliper framework provides workloads for average latency results.

After the implementation of the NIST ABAC architecture in a permissioned blockchain network, we implemented our policy review algorithms on the Hyperledger blockchain network. In section 4.4.2, we presented the evaluation of reading the Policy Information Ledger on the Hyperledger Fabric network and the response time for a policy review of various request sizes.

The are multiple tracks to further this work. The immediate effect is to incorporate the review of authorization and revocation of non-administrative policy into this work. The Policy Machine is robust and flexible to compose and combine multiple policies. An area for future work is to study the policy review in Policy Machine with multiple policy classes. Another extension of this work can include the policy review of delegations.

BIBLIOGRAPHY

[1] hyperledger-fabric.readthedocs.io release-2.2. `https://hyperledger-fabric.readthedocs.io/en/release-2.2/glossary.html?highlight=organization#organization`. Accessed: 2020-12-09.

[2] MAC A unified attribute-based access control model covering DAC and RBAC. X. jin, r. krishnan, and r. s. sandhu. In *DBSec 2012: Data and Applications Security and Privacy XXVI*, volume 7371, pages 41–55. Springer, Berlin, Heidelberg, 2012.

[3] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[4] Rejina Basnet, Subhojeet Mukherjee, Vignesh M Pagadala, and Indrakshi Ray. An efficient implementation of next generation access control for the mobile health cloud. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 131–138. IEEE, 2018.

[5] Elisa Bertino, Carolyn Brodie, Seraphin B Calo, Lorrie Faith Cranor, C Karat, John Karat, Ninghui Li, Dan Lin, Jorge Lobo, Qun Ni, et al. Analysis of privacy and security policies. *IBM Journal of Research and Development*, 53(2):3–1, 2009.

[6] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. Attribute transformation for attribute-based access control. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 1–8, 2017.

[7] Richard Gendal Brown, James Carlyle, Ian Grigg, and Mike Hearn. Corda: an introduction. *R3 CEV, August*, 1:15, 2016.

[8] Kyle Haefner Bruhadeshwar Bezawada and Indrakshi Ray. Securing home iot environments with attribute-based access control. In *Proc. of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18), New York, NY, USA*, pages 43–53. ACM Press, March 2018.

[9] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[10] Christian Cachin et al. Architecture of the hyperledger blockchain fabric. In *Workshop on distributed cryptocurrencies and consensus ledgers*, volume 310. Chicago, IL, 2016.

[11] Jason Crampton, Charles Morisset, and Nicol Zannone. On missing attributes in access control: Non-deterministic and probabilistic attribute retrieval. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 99–109, 2015.

[12] S. Gavrila D. Ferraiolo and W. Jansen. Policy machine: features, architecture, and specification. Technical Report 7987 Rev. 1, National Institute of Standards and Technology, Internal Report, 2015.

[13] Gopi Katwala David Ferraiolo, Serban Gavrila and Joshua Roberts. Imposing fine-grain next generation access control over database queries. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC '17), New York, NY, USA*, pages 9–15. ACM Press, March 2017.

[14] Serban Gavrila David Ferraiolo and Gopi Katwala. A system for centralized abac policy administration and local abac policy decision and enforcement in host systems using access control lists. In *Proc. of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18), New York, NY, USA*, pages 35–42. ACM Press, March 2018.

[15] Sabrina De Capitani di Vimercati, Pierangela Samarati, and Sushil Jajodia. Policies, models, and languages for access control. In *International Workshop on Databases in Networked Information Systems*, pages 225–237. Springer, 2005.

[16] Sheng Ding, Jin Cao, Chen Li, Kai Fan, and Hui Li. A novel attribute-based access control scheme using blockchain for iot. *IEEE Access*, 7:38431–38441, 2019.

[17] Chethana Dukkipati, Yunpeng Zhang, and Liang Chieh Cheng. Decentralized, blockchain based access control framework for the heterogeneous internet of things. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 61–69, 2018.

[18] Arjumand Fatima, Yumna Ghazi, Muhammad Awais Shibli, and Abdul Ghafoor Abassi. Towards attribute-centric access control: an abac versus rbac argument. *Security and Communication Networks*, 9(16):3152–3166, 2016.

[19] Maribel Fernández, Ian Mackie, and Bhavani Thuraisingham. Specification and analysis of abac policies via the category-based metamodel. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 173–184, 2019.

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

[21] Anna Lisa Ferrara, P Madhusudan, and Gennaro Parlato. Policy analysis for self-administrated role-based access control. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 432–447. Springer, 2013.

[22] Mikhail I Gofman, Ruiqi Luo, Ayla C Solomon, Yingbin Zhang, Ping Yang, and Scott D Stoller. Rbac-pat: A policy analysis tool for role based access control. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 46–49. Springer, 2009.

[23] Mikhail I Gofman, Ruiqi Luo, and Ping Yang. User-role reachability analysis of evolving administrative role based access control. In *European Symposium on Research in Computer Security*, pages 455–471. Springer, 2010.

[24] Hao Guo, Wanxin Li, Mark Nejad, and Chien-Chung Shen. Access control for electronic health records with hybrid blockchain-edge architecture. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 44–51. IEEE, 2019.

[25] Hao Guo, Ehsan Meamari, and Chien-Chung Shen. Multi-authority attribute-based access control with smart contract. In *Proceedings of the 2019 International Conference on Blockchain Technology*, pages 6–11, 2019.

[26] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162):1–54, 2013.

[27] Vincent C Hu and Karen Ann Kent. *Guidelines for access control system evaluation metrics*. Citeseer, 2012.

[28] Vincent C Hu, D Richard Kuhn, David F Ferraiolo, and Jeffrey Voas. Attribute-based access control. *Computer*, 48(2):85–88, 2015.

[29] Jingwei Huang, David M Nicol, Rakesh Bobba, and Jun Ho Huh. A framework integrating attribute-based policies into role-based access control. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 187–196, 2012.

[30] Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *IEEE Transactions on Parallel and Distributed Systems*, 22(7):1214–1221, 2010.

[31] Karthick Jayaraman, Mahesh Tripunitara, Vijay Ganesh, Martin Rinard, and Steve Chapin. Mohawk: Abstraction-refinement and bound-estimation for verifying access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 15(4):1–28, 2013.

[32] Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 41–55. Springer, 2012.

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

[34] Serban Gavrila David Ferraiolo Joshua Roberts, Gopi Katwala. Web Based Implementation of Policy Machine. `https://pm-master.github.io/pm-master/`, May 2020.

[35] J. van Deventer J. Eliasson J. Delsing K. K. Kolluru, C. Paniagua and R. J. DeLong. An aaa solution for securing industrial iot devices using next generation access control. In *2018 IEEE Industrial Cyber-Physical Systems (ICPS), St. Petersburg*, pages 737–742. IEEE, 2018.

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

[37] Wouter Kuijper and Victor Ermolaev. Sorting out role based access control. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 63–74, 2014.

[38] Jan Lánskỳ et al. Analysis of cryptocurrencies price development. *Acta Informatica Pragensia*, 5(2):118–137, 2016.

[39] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. A blockchain based approach for the definition of auditable access control systems. *Computers & Security*, 84:93–119, 2019.

[40] Andreas Matheus. How to declare access control policies for xml structured information objects using oasis'extensible access control markup language (xacml). In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, pages 168a–168a. IEEE, 2005.

[41] Samrat Mondal, Shamik Sural, and Vijayalakshmi Atluri. Security analysis of gtrbac and its variants using model checking. *computers & security*, 30(2-3):128–147, 2011.

[42] Satoshi Nakamoto. Re: Bitcoin p2p e-cash paper. *The Cryptography Mailing List*, 2008.

[43] R. Sandhu P. Biswas and R. Krishnan. Label-based access control: An abac model with enumerated authorization policy. In *Proc of the 2016 ACM International Workshop on Attribute Based Access Control*, pages 1–12. ACM Press, October 2016.

[44] Ray I. Pagadala V. Achieving mobile-health privacy using attribute-based access control. In *Foundations and Practice of Security. FPS*, volume 11358, pages 301–316. Springer-Cham, November 2018.

[45] James M. Shook Peter Mell and Serban Gavrila. Restricting insider access through efficient implementation of multi-policy access control systems. In *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats(MIST '16), New York,NY, USA*, pages 13–22. ACM Press, October 2016.

[46] Otto Julio Ahlert Pinno, André Ricardo Abed Grégio, and Luis CE De Bona. Controlchain: A new stage on the iot access control authorization. *Concurrency and Computation: Practice and Experience*, 32(12):e5238, 2020.

[47] Torsten Priebe, Wolfgang Dobmeier, and Nora Kamprath. Supporting attribute-based access control with ontologies. In *First International conference on availability, reliability and security (ARES'06)*, pages 8–pp. IEEE, 2006.

[48] V. M. Pagadala R. Basnet, S. Mukherjee and I. Ray. An efficient implementation of next generation access control for the mobile health cloud. In *Third International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 131–138. IEEE, 2018.

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

[50] Frank P Ramsey. On a problem of formal logic. *Proceedings of the London Mathematical Society*, 2(1):264–286, 1930.

[51] Silvio Ranise, Anh Truong, and Alessandro Armando. Scalable and precise automated analysis of administrative temporal role-based access control. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 103–114, 2014.

[52] Indrakshi Ray, Robert France, Na Li, and Geri Georg. An aspect-based approach to modeling access control concerns. *Information and Software Technology*, 46(9):575–587, 2004.

[53] Sara Rouhani, Rafael Belchior, Rui S Cruz, and Ralph Deters. Distributed attribute-based access control system using permissioned blockchain. *World Wide Web*, pages 1–28, 2021.

[54] F. Patwa S. Bhatt and R. Sandhu. An attribute-based access control extension for openstack and its enforcement utilizing the policy machine. In *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC-16), Pittsburgh, PA, USA*, pages 37–45. ACM Press, November 2016.

[55] Pierangela Samarati and Sabrina Capitani de Vimercati. Access control: Policies, models, and mechanisms. In *International School on Foundations of Security Analysis and Design*, pages 137–196. Springer, 2000.

[56] 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.

[57] Ravi Sandhu and HL EJC. Feinstein, and ce youman. *Role-based access control models*, pages 38–47, 1996.

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

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

[60] Amit Sasturkar, Ping Yang, Scott D Stoller, and CR Ramakrishnan. Policy analysis for administrative role-based access control. *Theoretical Computer Science*, 412(44):6208–6234, 2011.

[61] D. Servos and S. L. Osborn. HGABAC. Hgabac: Towards a formal model of hierarchical attribute-based access control. In *Foundations and Practice of Security FPS 2014*, volume 8930, pages 187–204. Springer Cham, April 2015.

[62] Daniel Servos and Sylvia L. Osborn. Current research and open problems in attribute-based access control. *ACM Computing Survey*, 49(4):45, February 2017.

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

[64] Farhan Patwa Smriti Bhatt and Ravi Sandhu. Abac with group attributes and attribute hierarchies utilizing the policy machine. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC '17), New York, NY, USA*, pages 17–28. ACM Press, March 2017.

[65] Ian Steadman. Wary of bitcoin? a guide to some other cryptocurrencies. *Ars Technica*, 11, 2013.

[66] Scott D Stoller, Ping Yang, C R Ramakrishnan, and Mikhail I Gofman. Efficient policy analysis for administrative role based access control. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 445–455, 2007.

[67] Emre Uzun, Vijayalakshmi Atluri, Shamik Sural, Jaideep Vaidya, Gennaro Parlato, Anna Lisa Ferrara, and Madhusudan Parthasarathy. Analyzing temporal role based access control models. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*, pages 177–186, 2012.

[68] Rick Kuhn Arthur R. Friedman Alan J. Lang Margaret M. Cogdell Adam Schnitzer Kenneth Sandlin Robert Miller Vincent C. Hu, David Ferraiolo and Karen Scarfone. Guide to

attribute based access control (abac) definition and considerations (draft). Technical Report 800 (2013), 162, NIST, 2013.

[69] Zhongyuan Xu and Scott D Stoller. Mining parameterized role-based policies. In *Proceedings of the third ACM conference on Data and application security and privacy*, pages 255–266, 2013.

[70] Ping Yang, Mikhail I Gofman, Scott D Stoller, and Zijiang Yang. Policy analysis for administrative role based access control without separate administration. *Journal of Computer Security*, 23(1):1–29, 2015.

[71] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE, 2005.

[72] Nan Zhang, Mark Ryan, and Dimitar P Guelev. Evaluating access control policies through model checking. In *International Conference on Information Security*, pages 446–460. Springer, 2005.

[73] Xiaoshuai Zhang and Stefan Poslad. Blockchain support for flexible queries with granular access control to electronic medical records (emr). In *2018 IEEE International conference on communications (ICC)*, pages 1–6. IEEE, 2018.

VITA

Sherifdeen Opeyemi Lawal was born in Kaduna in Nigeria. He went to Elementary and secondary school in Erin-Ile Kwara State, Nigeria. He received (2007) his Bachelor of Science in Electrical and Computer Engineering from the Federal University of Technology (FUT) Minna Niger State Nigeria. In 2013, He moved to the United States for graduate studies. He completed his Master of Science in Computer Engineering from the University of Texas at San Antonio (UTSA), San Antonio, Texas, in 2015. He subsequently enrolled in the doctoral program in Electrical Engineering at the Department of Electrical and Computer Engineering in Fall 2016 in collaboration with the Institute for Cyber Security Center for Security and Privacy Enhanced Cloud Computing at the University of Texas at San Antonio (UTSA), San Antonio, Texas.

His research interests are in the areas of privacy and access management for cloud and blockchain platforms. He has delved into application development and cloud automation toolchain.