3-12-2020 11:00 AM

Hierarchical Group and Attribute-Based Access Control: Incorporating Hierarchical Groups and Delegation into Attribute-Based Access Control

Daniel Servos
The University of Western Ontario

Supervisor
Osborn, Sylvia L.
The University of Western Ontario Joint Supervisor
Bauer, Michael
The University of Western Ontario

Graduate Program in Computer Science
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Abstract

Attribute-Based Access Control (ABAC) is a promising alternative to traditional models of access control (i.e. Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-Based Access control (RBAC)) that has drawn attention in both recent academic literature and industry application. However, formalization of a foundational model of ABAC and large-scale adoption is still in its infancy. The relatively recent popularity of ABAC still leaves a number of problems unexplored. Issues like delegation, administration, auditability, scalability, hierarchical representations, etc. have been largely ignored or left to future work. This thesis seeks to aid in the adoption of ABAC by filling in several of these gaps.

The core contribution of this work is the Hierarchical Group and Attribute-Based Access Control (HGABAC) model, a novel formal model of ABAC which introduces the concept of hierarchical user and object attribute groups to ABAC. It is shown that HGABAC is capable of representing the traditional models of access control (MAC, DAC and RBAC) using this group hierarchy and that in many cases it's use simplifies both attribute and policy administration. HGABAC serves as the basis upon which extensions are built to incorporate delegation into ABAC.

Several potential strategies for introducing delegation into ABAC are proposed, categorized into families and the trade-offs of each are examined. One such strategy is formalized into a new User-to-User Attribute Delegation model, built as an extension to the HGABAC model. Attribute Delegation enables users to delegate a subset of their attributes to other users in an *"off-line"* manner (not requiring connecting to a third party).

Finally, a supporting architecture for HGABAC is detailed including descriptions of services, high-level communication protocols and a new low-level attribute certificate format for exchanging user and connection attributes between independent services. Particular emphasis is placed on ensuring support for federated and distributed systems. Critical components of the architecture are implemented and evaluated with promising preliminary results.

It is hoped that the contributions in this research will further the acceptance of ABAC in both academia and industry by solving the problem of delegation as well as simplifying administration and policy authoring through the introduction of hierarchical user groups.

Keywords: Attribute-Based Access Control, ABAC, hierarchy, delegation, access control model, HGABAC, Hierarchical Group and Attribute-Based Access Control, HGAA, Hierarchical Group Attribute Architecture

Summary for Lay Audience

Traditionally, access control policies have been based on the direct assignment of permissions or roles to users based on the user's identity. For example, Alice is granted permission to use the printer or Bob is grated the role of *"Manager"* and mangers can view employee salaries. Attribute-Based Access Control (ABAC) is a new take on access control that is identityless (i.e. the identity of the user is unknown at the time of policy creation). Instead, ABAC bases access control decisions on the attributes of the users (e.g. age, year level, certificates, etc.), the environment (e.g. date/time, number of users on-line, etc.) and objects being access (e.g. author, date created, security level, etc.). These attributes are related by an access control policies, for example, *"if the user is 18 years old or older they can read a book with an adult rating"*.

Basing access control decisions on attributes allows for increased flexibility when creating policies and enables new users to be placed into the system without assigning permissions or roles manually beforehand. However, as ABAC is relatively new, there are a number of issues that must be resolved before ABAC can see wider acceptance outside of academia. These issues include, but are not limited to, a lack of a delegation model, no support for user and object groups and no single agreement on a standard formal model of ABAC. The goal of this thesis is to produce potential solutions to these problems and thus aid in the adoption of ABAC.

A new ABAC model, entitled Hierarchical Group and Attribute-Based Access Control (HGABAC), is introduced which adds user and object groups to ABAC. It is shown that these groups can help both simplify administration of ABAC systems and allow HGABAC to be backwards compatible with traditional identity based policies. A delegation model is added that allows users to delegate a number of their attributes to other users. This delegation ability is important in many real-world scenarios including continuing business functions when an employee is absent. Lastly, a supporting architecture is provided to fill in the gaps and act as a bridge between the theoretical HGABAC model and a real-world implementation.

Co-Authorship Statement

This thesis is an integration of four articles published by the author, Daniel Servos, under the supervision of Dr. Sylvia L. Osborn[1] and one under the supervision of Dr. Michael Bauer. Each publication comprises a single chapter of this thesis and summarizes the results of the research conducted by the author as part of their doctoral studies at the University of Western Ontario. The following chapters are based on these published works:

Chapter 2: Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017

I am the principal author of this peer-reviewed journal publication which surveys the current research and open problems in the area of attribute-base access control. I performed the literature search, wrote the manuscript, and identified the open problems discussed in the article. Dr. Osborn acted in a supervisory role, providing guidance on the methodology used and editing and reviewing the manuscript.

This literature survey was based on an earlier unpublished manuscript created by the author as part of the departments Topics Survey/Proposal (TSP) requirement and was reviewed by the author's TSP Examination/Advisory Committee (Dr. Michael Bauer, Dr. Hanan Lutfiyya and Dr. Sylvia L. Osborn).

Chapter 3: Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *The 7th International Symposium on Foundations and Practice of Security (FPS'2014)*, pages 187–204, November 2014

I am the principal author of this peer-reviewed conference paper which was part of The 7th International Symposium on Foundations & Practice of Security (FPS'2014) and published by Springer. This article introduces the HGABAC model, a formalized Attribute-Based Access Control (ABAC) model that introduces hierarchical concepts to user and object groups not yet found in existing ABAC models. I created and formalized the model, wrote the manuscript, and presented the work at the conference. Dr. Osborn acted in a supervisory role, providing guidance on the formalization of the model as well as editing and reviewing the manuscript.

Chapter 4: Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *The 9th International Symposium on Foundations and Practice of Security (FPS'2016)*, pages 320–328, October 2016

I am the principal author of this peer-reviewed conference paper which was part of The 9th International Symposium on Foundations & Practice of Security (FPS'2016) and published by Springer. This article introduces a number of possible strategies for incorporating dynamic delegation into ABAC models and discusses trade-offs associated with each strategy. I identified and described each strategy discussed in the paper, evaluated their trade-offs, categorized the strategies, wrote the manuscript and presented the work

[1]Dr. Osborn supervised this thesis until their death on May 23, 2018. Dr. Bauer subsequently took over supervision of the thesis.

at the conference. Dr. Osborn acted in a supervisory role, providing guidance as well as editing and reviewing the manuscript.

Chapter 5: Daniel Servos and Sylvia L Osborn. HGAA: An architecture to support hierarchical group and attribute-based access control. In *The Proceedings of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18)*, pages 1–12, March 2018

I am the principal author of this peer-reviewed workshop paper which was included in the proceedings of the third ACM workshop on attribute-based access control (ABAC'18). A workshop that was part of the larger ACM conference on data and application security and privacy (CODASPY 2018). This work introduced an architecture to support HGABAC that fills the gaps between model and real world implementation. I created the architecture, protocol and policy language described in the article, wrote the manuscript and presented the architecture at the workshop. Dr. Osborn acted in a supervisory role, providing guidance as well as editing and reviewing the manuscript.

Chapter 6: Daniel Servos and Michael Bauer. Incorporating off-line attribute delegation into hierarchical group and attribute-based access control. In *The International Symposium on Foundations and Practice of Security (FPS'2019)*, 2019. Also forthcoming publication in Springer's Lecture Notes in Computer Science LNCS

I am the principal author of this paper which was part of The 12th International Symposium on Foundations & Practice of Security (FPS'2019). It is also forthcoming publication in Springer's Lecture Notes in Computer Science (LNCS). This paper builds on the work introduced in previous chapters to provide a delegation model for HGABAC (Chapter 3) based on the User-to-User Attribute Delegation Strategy (Chapter 4) as well as extensions to the HGAA architecture (Chapter 5) and attribute certificate to support delegation. I created and formalized the model and extensions, wrote the manuscript, and presented the work at the conference. Dr. Bauer acted in a supervisory role, providing guidance on the formalization of the model as well as editing and reviewing the manuscript.

Acknowlegements

I would like to express my sincere gratitude to my supervisor and mentor, the late Dr. Sylvia Osborn, who passed away before the completion of my thesis. Her guidance, support and patience were invaluable to my research and graduate studies. I have been extremely fortunate to have an advisor as experienced and knowledgeable as Sylvia and I greatly appreciate the time she devoted to my research, publications and questions. I would also like to thank Dr. Michael Bauer for taking over supervision of my doctoral studies and severing on my TSP Examination/Advisory Committee.

Besides my supervisors, I would also like to acknowledge the contributions from the anonymous reviewers of my publications who's advice, recommendations and constructive criticism allowed me to improve my research and elevate the quality of my manuscripts. The insightful comments and feedback from the attendees of the FPS'14, FPS'16, FPS'19 and CODASPY 2018 conferences have also helped improve and shape the direction of my research.

My sincere thanks also goes to the members of my TSP, supervisory and examination committees for their comments and taking the time to read my work.

I must also express my gratitude to my wife, Jessica Servos, for her continued support, understanding and for proof reading of my publications, thesis and presentations.

I would also like to acknowledge TD Insurance Meloche Monnex for going above and beyond what was required of them after a fire destroyed my home and personal belongs including a number of computers, hard drives and texts that pertained to my research and publications. Their quick action allowed for the recovery of lost data, swift replacement of computing equipment and dealing with destroyed and damaged texts borrowed from the University library and repositories.

Financial support was provided by the Ontario provincial government, Western University, and private donor funding from the Robert E. Ferguson and Betty Spencer Ontario Graduate Scholarships.

Finally, I would like to thank the staff and faculty of the Western University Department of Computer Science for their continued support and understanding through a number of difficult circumstances.

Table of Contents

List of Figures

List of Tables

List of Examples

List of Appendices

List of Abbreviations

AA Attribute Authority. 122, 123, 130–132, 134

AAA Authentication, Authorization and Accounting. 95, 99

ABAC Attribute-Based Access Control. iv, 1–3, 7–24, 26–54, 56–58, 61, 64, 73, 76, 78–80, 84, 87, 89–91, 93, 95, 96, 99, 118, 119, 121–125, 136, 138–146, 163

ABAM Attribute-Based Access Matrix. 12, 18, 23, 49

ABCL Attribute-Based Constraints Specification Language. 49

ABE Attribute-Based Encryption. 10, 16, 56, 57, 78, 79

ABMAC Attribute-Based Multipolicy Access Control. 32, 57

ABNF Augmented Backus–Naur Form. 66, 67, 109

AC Attribute Certificate. 99, 101, 102, 104, 108–111, 116, 117, 122, 123, 130–132, 134, 136, 143–145, 147

ACL Access Control List. 55

ACM Association for Computing Machinery. iv, v

ANSI American National Standards Institute. 25

ARBAC Attribute and Role-Based Access Control. 42

ASN.1 Abstract Syntax Notation One. 102, 136

AST Abstract Syntax Tree. 113, 114, 116, 117, 144

BABAC Behaviours and Attributes-Based Access Control. 45, 46

BBAC Behaviour-Based Access Control. 35, 39, 45

CA-ABAC Class Algebra Attribute-Based Access Control. 27

CABAC Contextual Attribute-Based Access Control. 31

CD Can Delegate. 127

CODASPY Conference on Data and Application Security and Privacy. v, vi

CP-ABE Ciphertext-Policy Attribute-Based Encryption. 10, 56

CPL Common Policy Language. 24, 39, 44, 66

CPS Cyber-Physical System. 27

DAAC Delegatable Attribute-Based Anonymous Credentials. 138

DAC Discretionary Access Control. 2, 4, 6, 11, 13, 21, 24–26, 48, 54–58, 74, 76, 95, 119, 121, 141, 143, 145

DAC Delegated Attribute Certificates. 131, 132, 134

DAS Delegated Attribute Set. 126, 129–131, 134

FICAM Federal Identity, Credential, and Access Management. 12

FOL First Order Logic. 43

FPS Foundations & Practice of Security. iv–vi, 10, 55, 78, 118

FRBAC Flat Role-Based Access Control. 37, 40

GARBAC General Attribute-Based Role-Based Access Control. 42

GLIN Global Legal Information Network. 33

Grid_ABAC Grid Attribute-Based Access Control. 32

GT4 Globus Toolkit release 4. 32

GTRBAC Generalized Temporal Role-Based Access Control. 37, 41, 42

GURA Generalized URA Model. 139

HGAA Hierarchical Group Attribute Architecture. v, 12, 14, 54, 90, 92, 95, 96, 98, 99, 101, 111, 117, 119, 122, 123, 130–134, 136, 138, 142–148

HGABAC Hierarchical Group and Attribute-Based Access Control. iv, v, 10–14, 26, 54, 56–60, 63, 65–69, 73–76, 79, 84, 90–96, 99, 101, 102, 104, 108, 109, 116, 117, 119–122, 124–128, 130–132, 136, 139–148

HGPL Hierarchical Group Policy Language. 13, 14, 26, 66, 68, 76, 116, 120, 121, 126, 130, 131, 134, 136, 143, 144, 146, 147

HGPLv1 Hierarchical Group Policy Language Version 1. 67–69, 94, 95, 162

HGPLv2 Hierarchical Group Policy Language Version 2. 69, 109, 115, 116, 121, 134, 143, 144

ID Identifier. 3, 21, 108–110, 126–128, 134

INCITS InterNational Committee for Information Technology Standards. 25

IoT Internet of Things. 1, 147

IP Internet Protocol. 21, 127, 130

JSON JavaScript Object Notation. 92, 111

KP-ABE Key-Policy Attribute-Based Encryption. 10

MAC Mandatory Access Control. 2, 4–6, 11, 13, 21, 23–26, 48, 54–58, 74, 76, 95, 119, 121, 141, 143

MPABAC Multiple-Policy Attribute-Based Access Control. 30, 31

NIST National Institute of Standards and Technology. 2, 6, 7, 12, 39, 44, 47, 75, 96

NLACP Natural Language Access Control Policy. 140

OASIS Organization for the Advancement of Structured Information Standards. 10

ORM Object Relational Mapper. 111

OWL Web Ontology Language. 38, 41

OWL-DL OWL Description Logic. 38, 41

PAP Policy Administration Point. 95, 96, 99

PBAC Policy-Based Access Control. 53

PDP Policy Decision Point. 95, 96, 99

PEP Policy Enforcement Point. 95, 99

PFRBAC Parameterized Flat Role-Based Access Control. 40

PGP Pretty Good Privacy. 51

PIP Policy Information Point. 95, 96, 99

PKI Public Key Infrastructure. 96, 102, 111

PM Policy Machine. 24, 25, 96, 139, 140

PRBAC Parameterized Role-Based Access Control. 20, 36, 40, 44

PRP Policy Retrieval Point. 95, 99

RABAC Role-centric Attribute-Based Access Control. 35, 44

RB-RBAC Rule-Based Role-Based Access Control. 40, 42

RBAC Role-Based Access Control. 2, 3, 6, 7, 9, 11–13, 18, 20, 21, 23–26, 34–44, 46, 48–58, 64, 74–76, 79, 83, 95, 102, 119, 121, 138, 140, 141, 143, 146–148

RBM Role-Based Management. 37, 40

RDF Resource Description Framework. 117

RDFS Resource Description Framework Schema. 117

ReBAC Relationship-Based Access Control. 35, 44

RFID Radio-frequency identification. 31

rHGABAC Restricted Hierarchical Group and Attribute-Based Access Control. 91, 96, 139, 140

SABAC Semantic-Aware Attribute-Based Access Control. 33

SAML Security Assertion Markup Language. 10, 16, 20, 33, 37, 42, 87, 95, 96, 117, 147

SIN Social Insurance Number. 3

SNS Social Network Systems. 44–46

SOA Service-Oriented Architectures. 33

SoD Separation of Duties. 6, 49, 142, 146

SPARQL SPARQL Protocol and RDF Query Language. 38, 41

SQL Structured Query Language. 117

SSO Single Sign-On. 95

SWRL Semantic Web Rule Language. 38, 41

T-ABAC real-Time Attribute-Based Access Control. 27, 30

TBAC Task-Based Authentication Control. 39, 46

TLS Transport Layer Security. 111

TP-Functions Token-Provisioning Functions. 23

TP-Graph Token-Provisioning Graph. 23

TSP Topics Survey/Proposal. iv, vi

UID Unique Identifier. 104, 111, 132

URA User-Role Assignment. 139

URI Uniform Resource Identifier. 69, 92, 101, 102, 109, 122, 143

URL Uniform Resource Locator. 102, 104

UURAC User-to-User Relationship-Based Access Control. 39, 44, 45

WS-ABAC Web Service Attribute-Based Access Control. 32, 33, 57

X-GTRBAC XML-based Generalized Temporal Role-Based Access Control. 37, 42

XACML eXtensible Access Control Markup Language. 3, 9, 10, 16, 20, 21, 24, 28, 29, 31–33, 39, 47, 52, 56, 57, 66, 68, 76, 86, 95, 96, 101, 117, 118, 141, 146, 147

XML eXtensible Markup Language. 10, 36, 40, 42, 52, 56, 95

XPath XML Path Language. 36, 37

Chapter 1

Introduction

1.1 Motivation

The early 2010s saw rise to a popular trend of moving computing infrastructure from traditional localized equipment to decentralized service-orientation or cloud-based platforms. While *"the cloud"* offered much potential for *"on demand"* dynamic scaling of applications and resources through virtualization, it created a need for new models of access control to match the remote and agile nature of the computing environment. Attribute-Based Access Control (ABAC) was suggested as a possible solution to providing a unified, but flexible access control policy for this new paradigm [Hur and Noh 2011; Lang et al. 2009; Yuan and Tong 2005]. In more recent years, interest has grown in applying Attribute-Based Access Control (ABAC) to emerging computing areas such as Internet of Things (IoT)[Bhatt 2018], smart connected vehicles [Gupta et al. 2019], health care[Servos 2012], big data[Gupta et al. 2018] and other less traditional distributed systems.

ABAC, unlike more traditional models of access control, allows for the creation of access policies based around the existing attributes of the users and objects in the system, rather than the manual assignment of roles, ownership or security labels by a system administrator. There are several situations, including cloud computing, where this would be beneficial, removing the need for manual intervention when authorizing users for certain roles or security levels; simplifying administration in complex systems with a large number of users as well as creating the possibility of automating access control decisions for remote users from foreign systems.

While some research has focused on adding attribute-like concepts to existing models of access control[Al-Kahtani and Sandhu 2002, 2003], little work has been successful in creating a generally agreed upon formalized model or standardization of "pure" Attribute-Based Access Control. This shortage of formalized foundational ABAC models is limiting both access control research (forcing attribute related work to be highly informal and ill defined) and industry (causing a lack of ABAC standardization needed for implementing secure attribute-based systems). This problem has not gone unnoticed by government and standards organizations, leading to calls for increased development of ABAC systems and research towards foundational models by the Federal Chief Information Officers Council[CIO Council 2011], the National Institute of Standards and Technology (NIST)[Hu et al. 2013], and even as part of the White House's National Strategy for Information Sharing and Safeguarding[Office of the Press Secretary 2012].

A formalized foundational model of ABAC will open new research opportunities in access control as well as lead the way to standardization for industry use; however, ABAC's relative infancy compared to the traditional models (namely MAC, DAC, and RBAC) leads to a number of other related research problems (discussed in Chapter 2 Section 2.4). These problems include but are not limited to: lack of a delegation and administration model, limited work towards hierarchical models, questionable auditability and the complexity of enforcing separation of duty constraints. Solving these problems is critical to support the adoption of ABAC systems in real world scenarios and move beyond the theoretical level.

1.2 Background

1.2.1 Basic Definitions

The following are commonly used definitions in access control research and the remainder of this document:

Object: A logical object on which access control is desired (sometimes referred to as a **resource**). Examples include but are not limited to: file system objects, databases, physical resources (e.g. printers, scanners, doors, etc.), network resources, other access control entities (e.g. subjects or roles) and physical objects (e.g. library book).

Subject: An entity requesting access to the system (also referred to as a **user** or **requestor**). This may be the actual human user of the system, a session or process that performs requests on the user's behalf or a completely automated non-human program or process (a non-person entity).

Operation: A process, command, or program that may be performed on an object (also called **actions** or **access modes**). Examples include but are not limited to; read, write, delete, execute (e.g. run a program), grant (grant ownership or permission on), use (e.g. use a printer or open a door) or a custom operation like checkout a book (e.g. for a system granting permission to checkout library books).

Permission: The authorized actions in a system that are commonly defined as a paring of an operation with the object it is allowed to be performed on. Permissions are also commonly referred to as **privileges**, **rights**, **authorizations** or **entitlements** but may have slightly differing definitions in some works.

Policy: (or **Security Policy**, **Access Control Policy**, **Access Policy**, **Policy Rule**, etc.) are the high level rules system administrators put in place to govern allowable behaviour. These policies may take the form of written rules in well defined policy languages (e.g. XACML) or more abstract configurations such as defining and assigning roles in RBAC.

Administration: The act of creating, defining, editing, configuring or maintaining access control policies.

Authentication: The act of verifying that a subject is actually who they claim to be (normally on the basis of some credentials provided by the subject).

Credential: Proof of a subject's identity, qualification, competence, or authority. Can take the form of a user name, password pair, signed certificate issued by some authority, verifiable statements made by other subjects, etc.

Identity: A unique identifier or aspect of a subject (e.g. a user's full name, employee ID, SIN, etc.). This is distinct from common attributes of a subject that are not necessarily unique on their own (e.g. age, role, job title, student type) but may form a unique identity when combined.

Attribute: A trait or aspect of an access control entity, the environment or any part of the system for which access control is being applied. Possible subject attributes include age, name, home address, or job title. Possible object attributes include author, creation date, last modification date, or patient (for a health record). Possible environment attributes might include current time, day of the week, number of users logged in, or free space. See Section 1.2.3 for more details.

Constraint: A limitation or restriction placed on part of an access control system. For example, a constraint may restrict a user from having both permissions required to create a loan application and approve a loan application or a system may be constrained to only allow a fixed number of roles to be activated in a single session. Additionally, constraints can be made based on the current state of the system's environment (e.g. only allow role/permission activation during set times or days of the week); however, this should be considered distinct from the permission granting policies in ABAC that grant permissions based on the current value of attributes of the environment and other access control entities (i.e. one is restricting permissions and one is granting permissions).

Group: A collection of access control entities that may be assigned or related to other entities as a whole. For example, an object group might be a collection of resources that may be paired with a single operation to form a permission containing multiple objects.

Role: A set of permissions assigned together as a group to other access control entities (namely users). Often access control roles are representative of real roles or job titles found in an organization and may have a similar hierarchical representation. Used in RBAC and hybrid ABAC models, see Section 1.2.2.3.

Session: A logical construct, often only temporary in existence, that makes access control requests on a system in place of a user. In many models of access control a session will only be granted a subset of a user's permissions in an effort to help enforce the principle of least privilege.

Definitions with multiple corresponding words (e.g "user" and "subject" or "object" and "resource") are used interchangeably in this document and refer to the above definitions unless otherwise noted.

1.2.2 Traditional Access Control

1.2.2.1 Discretionary Access Control

In Discretionary Access Control (DAC)[Lampson 1974; Griffiths and Wade 1976], access is granted to users directly via an access matrix[Harrison et al. 1976]. In most cases this matrix, A, takes the form of $A[s_i, o_j] = m_{i,j}$ where s_i is a subject from the set of all subjects that have access to a given system, o_j is an object from the set of all objects or resources protected by the system and $m_{i,j}$ is the set of access modes under which the subject may access the object (e.g. the "read" access mode might grant the subject permission to view the contents of the object). Granting permissions on an object (i.e. assigning access modes to the user, object pair in the access matrix) is left to the discretion of the object's "owner", either an individual user or group that has been previously assigned ownership of the object (often represented through an "owns" access mode in the access matrix and assigned to the initial creator of an object by default).

While the simplicity of this model makes it ideal for cases like file system access control in operating systems (e.g. Unix file permissions), it is insufficiently flexible or constrainable for systems with more complex access control requirements. In many cases it is desirable that access control decisions are centrally controlled and made on the basis of organizational policies or structure, rather than left to the discretion of individual users of the system. In terms of flexibility, DAC lacks the expressiveness to deal with more complex access policies that occur in real world organizations such as "only accountants can access client files", "only users with top security clearance can read documents labelled top secret" or "tellers can only make changes to client's accounts during week days from 9AM to 5PM". Instead, DAC relies upon the manual tagging of object,subject pairs with access modes by individual owners, making it unsuitable for such scenarios.

1.2.2.2 Mandatory Access Control

Unlike DAC, Mandatory Access Control (MAC) enforces centrally controlled access policies defined not at the discretion of the user but by an access control administrator. This facilitates a single organization-wide security policy that applies to all subjects including the processes they execute. The Bell-LaPadula model[Bell and LaPadula 1973] is perhaps the most well known MAC model in the access control literature, which takes a lattice-based approach to enforcing multilevel security using security labels. In the Bell-LaPadula model, subjects and objects are tagged with clearances and security levels respectively, both comprised of a classification (e.g. "Top Secret"), C, and a set of categories (e.g. {"NSA, NATO, CIA"}), S, such that labels

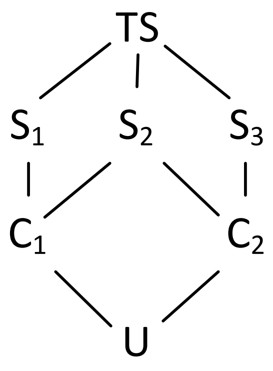


Figure 1.1: Example security lattice in the Bell-LaPadula[Bell and LaPadula 1973] model.

are defined as $L = (S, C)$. Classifications are totally ordered (e.g. "Top Secret" >"Secret" >"Classified" >"Unclassified") and a security level, $L_1 = (S_1, C_1)$, is considered to dominate another level, $L_2 = (S_2, C_2)$, if and only if $C_1 \geq C_2$ and $S_2 \subseteq S_1$. This partial ordering of security levels is represented by a mathematical lattice such as the example lattice shown in Figure 1.1.

Access control is established through the following "Simple Security" read property and either a "Liberal *" or "Strict *" write property (where $L(x)$ denotes the secularity label of the access control entity):

Simple Security Property: A subject, s, may read a given object, o, if $L(s) \geq L(o)$. That is to say that a subject may read the contents of an object if their security clearance dominates the security level of the object, restricting users to reading at their own level or lower in the lattice.

Liberal *-Property: A subject, s, may write to a given object, o, if $L(s) \leq L(o)$. That is to say that a subject may write to an object if the subject's clearance is dominated by the security level of the object, restricting users to writing to their own level or higher in the lattice.

Strict *-Property: A subject, s, may write to a given object, o, if $L(s) = L(o)$. That is to say that a subject may write to only objects of an equivalent security level.

If the example lattice from Figure 1.1 is used, a subject operating at a S_1 clearance level could read only objects labelled S_1, C_1 or U (all dominated by S_1) and write only to objects labelled S_1 or TS (assuming the liberal *-property is in use). If instead, the strict *-property is used, such a subject would be limited to writing to only objects of the same security level (S_1). In most systems, users are permitted to operate at any clearance level so long as that clearance level is dominated by the clearance label assigned to the user.

Although this style of MAC satisfies many of the requirements of the high security systems for which it was originally designed (namely for military and government use), it is often inadequate for more open and less static environments. The Bell-LaPadula model requires all subjects and objects to be both known and tagged with security levels before system creation and gives few options for reconfiguring the system at run time. While policies based on security levels are possible, more fine grained policies like "objects labelled Top Secret can only be accessed from a secure network" are still not possible using the model alone.

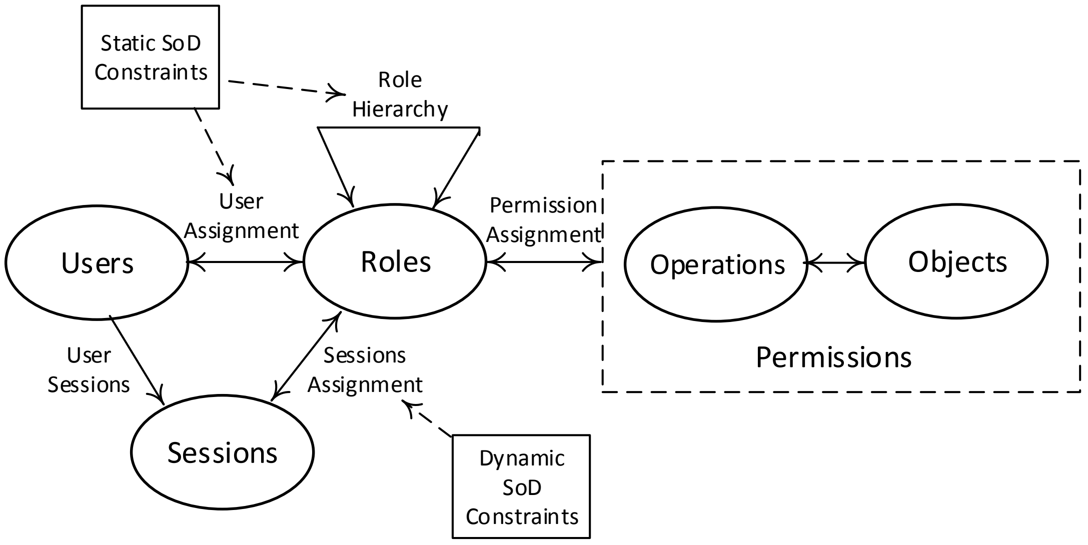


Figure 1.2: NIST RBAC Model[Ferraiolo et al. 2001].

1.2.2.3 Role-Based Access Control

In Role-Based Access Control (RBAC)[Ferraiolo et al. 2001; Nyanchama and Osborn 1999], access control decisions are based on a user's assigned role in an organization rather than directly assigned permissions or security labels. Roles are assigned permissions (object, operation pairs) that grant access to perform a specified operation on a specific object (or set of objects) and users are assigned to roles that represent their position in an organization. Users are able to activate one or more roles for which they are a member in a given session and all access decisions for that session are derived from the resulting set of permissions obtained from the active roles. One of the most accepted models of RBAC is the NIST RBAC model[Ferraiolo et al. 2001], shown in Figure 1.2, that includes a role hierarchy and constraints to enforce Separation of Duties (SoD). In the NIST model, the role hierarchy allows parent roles to inherit the permissions of their children and child roles to inherit the users of their parents. This allows RBAC to further model the hierarchical structure of most organizations. SoD constraints are divided into static SoD, where subjects are prohibited from being assigned conflicting roles, and dynamic SoD, where subjects are prohibited from activating conflicting roles in the same session.

While RBAC provides a more generalized model than MAC or DAC (and in fact can emulate both models effectively [Osborn et al. 2000]), it falls short in cases where users and their respective roles in the system are poorly defined or even unknown before access requests take place. As RBAC requires the assignment of users to roles (often manually) before users may access a system, it is unsuited for domains in which user's identities may be unknown or only determined at the time of access (such is commonly the case in service oriented architectures like web service based implementations). Additionally, access control polices in basic RBAC models (like the NIST model) are mostly limited to form of "if a user is assigned a role X they are granted the set of permissions Y"; however, this is insufficiently flexible for many real world scenarios. For example, a bank may only permit an employee with the role "teller" to access client accounts during set times of the day and week or a hospital may wish to constrain a user with the role “patient” to only viewing medical records in which they are described as the patient. Both of these cases would be difficult, if not impossible, to model in NIST style RBAC. While many extensions to RBAC do exist, which expand the flexibility of access control polices (such as those described in Chapter 2 Section 2.3.2) they often only concentrate on adding a single new feature (e.g. time or location based policies, parameterized roles, automated role assignment, etc.) rather than creating a new, more generalized model.

1.2.3 Attribute-Based Access Control

Attribute-Based Access Control (ABAC) is an emerging form of access control that is starting to garner interest in both recent academic literature and industry application. While there is currently no single agreed upon model or standardization of ABAC, there are commonly accepted high level definitions and descriptions of its function. One such high level description is given in National Institute of Standards and Technology (NIST)’s recent publication, a *“Guide to Attribute-Based Access Control (ABAC) Definition and Considerations”*[Hu et al. 2013]:

> ***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.*

Rather than basing access control decisions on a user’s identity like the traditional methods, ABAC bases access control around the attributes of access control entities. These attributes are often classified into one of the following categories:

User Attributes: Attributes of the subjects of the system. May include attributes like age, name, office number, job title, role, security clearance, home address, date hired, trust level (e.g. how trusted the user is by the system), etc.

Object Attributes: Attributes of the resources of the system. May include attributes about the meta-data related to the object such as author, date created, last modified, size, file type, security level, etc., or the contents of the object such as patient name (e.g. for health records), student number (e.g. for student records), title of chapter 1, etc.

Environmental Attributes: Attributes derived from the current state of the system’s environment. For example, current time, day of the week, number of users logged in, free space, CPU usage, etc.

Connection Attributes: Attributes that only apply to the current session of a user. For example, IP address, physical location (e.g. for mobile systems), session start date/time, current session length, host name, number of access requests made, etc.

Administrative Attributes: Configuration attributes that apply to the whole system and are either manually set by an administrator or by some automated process. These could include

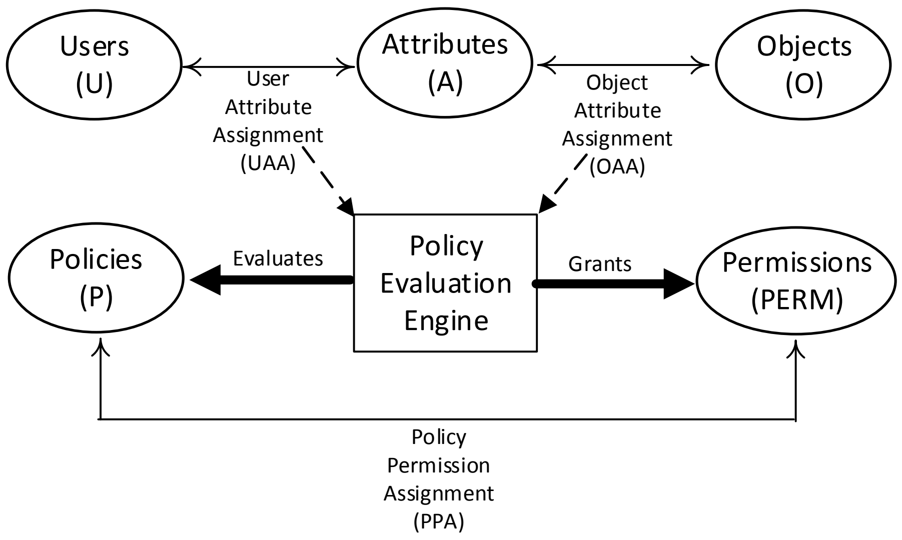


Figure 1.3: Core ABAC model. Thin solid arrows denote many-to-many relations, thick solid lines denote relation with policy engine and doted lines denote information used by the policy engine to evaluate a given policy. Ovals represent ABAC model elements.

a threat level (e.g. different polices could be used depending on if the system was likely to be attacked or not), minimum trust level (e.g. the minimum amount of trust required for a user to access the system), maximum session length (e.g. the maximum allowable length of a session), etc.

Ideally, these attributes are all properties of the elements in the system and do not need to be manually entered by administration (e.g. many of the attributes about an object come from its meta-data). Access policies can be created using policy languages, limiting access to certain resources or objects, based on the result of a Boolean statement comparing attributes, for example `user.age >= 18 OR object.owner == user.id`. This allows for flexible enforcement of real world policies, while only requiring knowledge of some subset of attributes about a given user (as opposed to knowing their identity and to what roles or permissions they have been manually assigned).

1.2.3.1 Core ABAC Model

This section gives a description of a simplified ABAC model based on common elements found in most ABAC models. While each ABAC model tends to formalize the elements of ABAC in a slightly different way, the following are the most common elements of an ABAC system and are present in most models:

Users (U): The set of all users that may access the system. Note that this set may not necessarily be finite as not all users are known at creation time (something that is common in service oriented architectures and systems involving information sharing across organizational boundaries).

Objects (O): The set of all objects protected by the system.

Attributes (A): The set of all attributes (given by a unique name) in the system. In some models, attributes also have a type associated with them or are subdivided into categories based on the access control entity to which they can be applied.

Permissions (PERM): The set of all possible permissions that may be granted to users. In some models, permissions consist of object, operation pairs similar to permissions in RBAC, but this is not necessarily required. In other models permissions are policy and operation pairs, that grant access to execute the operation on any object that fulfils the policy.

Policies (P): The set of all policies that govern access in the system. Normally these policies are written in a policy language and in some way related to permissions they grant.

These basic elements are assigned attributes and related through the following relations (shown in Figure 1.3):

Users Attribute Assignment (UAA): The assignment of attributes onto users. This may take the form of $\{a \in A, u \in U, values\} \in UAA$, that is to say that each element of UAA is a triple containing an attribute name from the set of attributes (A), a user from the set of users (U) and a set of values assigned to the given user and attribute pair. For example, if a user, u_1, was assigned an "age" attribute with the value of 28, the entry in UAA would look like $\{\text{"}age\text{"}, u_1, \{29\}\}$. Alternatively, if an user, u_2, was assigned a "supervises" attribute that contains the set of other users they supervise (in this case u_1 and u_3), the entry in UAA would look like $\{\text{"}supervises\text{"}, u_2, \{u_1, u_2\}\}$.

Object Attribute Assignment (OAA): The assignment of attributes onto object. This may take the form of $\{a \in A, o \in O, values\} \in UAA$, and works in the same way as UAA but with objects.

Policy Permission Relation (PPR): The relationship between policies and the permissions they grant. This may take the form of $\{p \in P, perm \subseteq PERM\} \in PPR$. This assignment is often formulated differently or not at all in many models depending on how their policy language works (e.g. the language itself may specify the permission set granted).

Policies in the PERM set are commonly Boolean statements involving attributes and constants such as `user.age >= 18` (grants access if the user is 18 or more years of age) or `user.id == object.author` (grants access if the user is the author of the file). When an access request is made by a user it is evaluated against the set of policies (P) given the assigned attributes of the user making the request and the object being requested. In many models, access requests are not conducted directly by the user but indirectly through a session that may contain a subset of the user's attributes. A comprehensive review of existing ABAC models is given in Chapter 2.

1.2.3.2 Policy Language Standards

A critical component of ABAC, although not strictly part of the ABAC model, is the access control policy language used to define policy rules for a system. These languages, while not models in them selves (as is sometimes erroneously implied), are either generic access control language standards (such as XACML) or languages created specifically for use with a single model. eXtensible Access Control Markup Language (XACML)[Godik et al. 2002], a standard created by the Organization for the Advancement of Structured Information Standards (OASIS), is one of the most frequently referenced works in ABAC literature. XACML is a XML based access control policy language that is notable for its support of attribute-based policies and used in multiple access control products. Similarly, Security Assertion Markup Language (SAML)[Hughes and Maler 2005], also developed by OASIS, provides a standardized markup language and protocol for exchanging authorization and authentication information between parties which supports attributes for authentication.

1.2.3.3 Attribute-Based Encryption

Another related but distinct research area from ABAC is Attribute-Based Encryption (ABE), where objects are encrypted based on attribute-based access policies. ABE mainly consists of Key-Policy Attribute-Based Encryption (KP-ABE)[Goyal et al. 2006] or Ciphertext-Policy Attribute-Based Encryption (CP-ABE)[Bethencourt et al. 2007; Servos et al. 2013] based encryption ciphers. In KP-ABE an object is encrypted with a set of attributes related to the object which must pass a policy embedded in a user's key for decryption to proceed. CP-ABE is the reverse of KP-ABE, using an attribute-based policy to encrypt an object and having a user's key consist of a set of attributes relating to that user. While ABE, much like XACML and SAML, lacks any kind of formal ABAC model and has rather simplified access policies, it does provide an interesting means of enforcing ABAC policies outside of the security domain they originate in. There are several examples of ABE being used for such in recent literature[Hur and Noh 2011; Wang et al. 2010; Servos 2012; Yu et al. 2010], particularly for securing web and cloud based services.

1.3 Goals & Contributions

The main purpose the research contained in this thesis is to create a formalized model of ABAC based in set theory which can act as a foundational model for both future research and standardization of ABAC for industry applications. Such a model would include both informal and formal definitions, a policy language for ABAC, example implementation and several previously overlooked aspects of ABAC, including delegation, support for hierarchical access control structures, an encompassing architecture for real world use and a set of suporting protocols and formats for dealing with attributes. As a first step towards this goal, a new model of hierarchical ABAC, entitled Hierarchical Group and Attribute-Based Access Control (HGABAC), was created (Chapter 3 and was presented in the refereed Symposium on Foundations & Practice of Security (FPS) conference (published in springer lecture notes in computer science[Servos and Osborn 2014]). This work presents a formal model of ABAC that is notable in part due to the addition of hierarchical user/object groups that allow for more simplistic and flexible administration of access control policies. This contribution serves as a foundational model upon which extensions and frameworks are built to support delegation and administrative functions in later chapters (Chapters 5 and 6).

1.3.1 Goals

The main goal of the research contained in this thesis is to provide potential solutions to a number of open problems identified in ABAC research (as identified in Chapter 2 Section 2.4). More specifically the following issues are addressed:

Hierarchical ABAC: At the time of creating the HGABAC model (Chapter 3), no *"pure"* models of ABAC supported hierarchical concepts similar to those found in modern RBAC models. One aim of the HGABAC model is to introduce hierarchical concepts into ABAC via attribute user and object group hierarchies in which attributes and their assigned values are inherited from parent groups. This aids in both administrative tasks and in increasing the flexibility of possible ABAC policies (including allowing for new methods of emulating the traditional models). HGABAC formalizes the concepts of hierarchical attributes groups and demonstrates their versatility in modelling DAC, MAC and RBAC.

Representing the Traditional Models: A key feature of next generation access control models is the ability to emulate policies of existing models. Just as RBAC was shown to be capable of representing and enforcing DAC and MAC based policies[Osborn et al. 2000], the next generation of ABAC models must have the same capability (i.e. enforcing DAC, MAC and RBAC policies). While some work towards representing the traditional models in ABAC has been conducted (namely that done by X. Jin et al.[Jin et al. 2012]), there is still much left to be done in way of exploring and evaluating alternative representations. HGABAC takes a further step by exploring representation using hierarchical groups (as opposed to attributes that contain partially ordered sets as in [Jin et al. 2012]).

Support for Distributed Systems: Attribute-based policies have a number of advantages in distributed environments, particularly in cases where authority is federated among distinct security domains with their own users, policies and resources that act somewhat independently. In such cases, one domain may have little or no knowledge of the users or their roles in another domain and limited ability to communicate with other domains directly (e.g. due to communication bottlenecks or parts of systems being accessible only in an *"off-line"* manner). The identityless nature of ABAC is advantageous in such scenarios as users can be assigned permissions via their attributes and properly placed in the foreign domain with no knowledge of the user's identity or role in their home domain. While this aspect of ABAC is one of its strongest features, many ABAC models, frameworks and architectures assume that all services are on a single local system or at least within a single security domain (with all components being trusted and able to freely communicate). One key goal of this research is to ensure the proposed ABAC model and supporting architectures are compatible with distributed and federated systems. In part, this entails: having a name-space that uniquely identifies access control elements (attributes, users, etc.) across independent domains; securely sharing users, attributes and permissions between separate security domains; limiting required communication between domains to avoid bottlenecks and scalability issues; and providing a means of authenticating with services in an *"off-line"* manner (i.e. without having to connect to a third party). These issues are largely addressed in Chapter 5 and updated to support delegation in Chapter 6.

Supporting Architecture: In many cases, an ABAC model is not enough on its own to implement a comprehensive real-world access control system. A support architecture, protocols and system for storing and sharing attributes is required to answer questions like *"who assigns the attributes?"*, *"how are attributes shared with each party?"*, *"how does the user provide proof of attribute ownership?"*, *"where and how are policies evaluated?"*, *"how will the model scale in real-world use?"*, etc. Chapter 5 presents Hierarchical Group Attribute Architecture (HGAA), a potential answer to these questions.

Delegation Model: At present, little research has be done towards creating a formal model of delegation for ABAC. This commonly desired access control feature is critical for ABAC acceptance but must be carefully considered to avoid introducing new security and privacy issues. Potential strategies for creating such a model are discussed in Chapter 4 in addition to security and privacy implications of each method. One strategy, User-to-User Attribute Delegation, is formalized in Chapter 6 and a support architecture is outlined.

1.3.2 Contributions

There has been a growing demand from both government and industry for more research and development into ABAC systems that demonstrates both the need for and significances of such works. The Federal Identity, Credential, and Access Management (FICAM) Roadmap and Implementation Plan v2.0[CIO Council 2011] published by the Federal Chief Information Officers Council in the United States recommends ABAC for *"promoting information sharing between diverse and disparate [federal] organizations"*. This recommendation is further strengthened by the National Strategy for Information Sharing and Safeguarding[Office of the Press Secretary 2012] making the implementation of the FICAM roadmap a priority across federal networks in all security domains. This has lead the National Institute of Standards and Technology to start initiatives towards formalizing and standardising ABAC, including their recent guide to ABAC[Hu et al. 2013]. Despite this demand and a number of efforts to create ABAC models, there is still no single standardized description of ABAC beyond some commonly agreed upon high level definitions.

While a large number of ABAC models have already been developed (as evident by the extensive list of models discussed in Chapter 2), the research contained in this thesis is novel in a number of ways. Firstly, seldom discussed aspects of ABAC such as hierarchy, groups, delegation, function in *"off-line"* and distributed environments, and supporting architectures and frameworks are developed that have been largely ignored in recent literature. These aspects are critical for real world adoption of ABAC systems and are required for providing a complete access control solution. Secondly, while there are many ABAC models targeted at specific use-cases (e.g. cloud computing, web services, etc.), there are relatively few that are *"pure"*, generic, formal and complete. Only two of the reviewed models in Chapter 2 (excluding HGABAC) would meet these criteria, one of which (ABAM[Zhang et al. 2005]), is not an identityless solution. Such a model is required to serve as a foundational model for ABAC research, much like how the NIST RBAC model[Ferraiolo et al. 2001] provided a generic formalized model to enable both RBAC research and industry use. The proposed HGABAC model in Chapter 3, could potentially be a first step towards such a foundational model and has already seen some acceptance and use by other researchers (as discussed in Section 7.1).

1.4 Outline & Overview

The remainder of this thesis is divided into five chapters following the integrated-article format[1]. Chapters 2 to 5 contain content from articles published by the author during the course of researching this topic with minor additions, corrections and connecting information. A statement of co-authorship can be found at the beginning of this thesis (page iv). Article content is reproduced with the permission of the appropriate copyright holders (details given in Appendix A). An overview of each chapter follows:

Chapter 2: Current Research and Open Problems in Attribute-Based Access Control

Chapter 2 provides a literature review of current (at the time of writing) research in the area of ABAC. A taxonomy of ABAC models is presented that aids in classifying and organizing the many ABAC models, frameworks and architectures that are frequently proposed in recent publications. Open problems for ABAC not yet addressed by the literature to date are identified and possible directions for future work related to these problems are given. A number of these open problems, namely lack of foundational models (Chapter 2 Section 2.4.1), representing the traditional models (Chapter 2 Section 2.4.2), need for hierarchical ABAC (Chapter 2 Section 2.4.3), delegation (Chapter 2 Section 2.4.6), and attribute storage and sharing (Chapter 2 Section 2.4.7) form the motivation for the research detailed in the subsequent chapters.

Chapter 3: HGABAC: Towards a Formal Model of Hierarchical ABAC

Chapter 3 outlines a new formal model of ABAC, entailed Hierarchical Group and Attribute-Based Access Control (HGABAC), that incorporates hierarchical user and object attribute groups. A novel concept not yet seen in the ABAC literature at the time. Additionally, an accompanying attribute-based policy language, HGPL, is introduced as well as a new type of attribute, the *"administrative attribute"*. It is shown that policies supported by HGABAC and HGPL are flexible enough to emulate and represent the traditional access control models (MAC, DAC, and RBAC) and that the new hierarchy can reduce complexity and simplify administration of attributes. The ABAC model and policy language presented in this chapter forms the basis for the work in subsequent chapters, each chapter adding to the capabilities of the HGABAC model, or developing supporting architectures and features for HGABAC.

Chapter 4: Strategies for Incorporating Delegation into ABAC

Chapter 4 discusses potential strategies for incorporating delegation into ABAC, namely HGABAC. Possible strategies are created by combining different *"delegation components"* (different delegators, delegatees, and delegatable components) to form unique theoretical delegation models. Each strategy is evaluated and categorised based on shared weaknesses and advantages. Detailed examples of how each category of delegation strategy would work but formalizing and implementation of the models is left to future work (e.g. the model of User-to-User Attribute Delegation defined in Chapter 6).

[1]As described by the Western Graduate & Postdoctoral Studies regulations and requirements for thesis content (Regulation 8.3.1).

Chapter 5: HGAA: An Architecture to Support HGABAC

Chapter 5 seeks to fill in the gaps left by the HGABAC model and provide a supporting architecture to implement HGABAC based systems. An access control architecture, entitled Hierarchical Group Attribute Architecture (HGAA), is proposed which defines how Attribute Stores, Attribute Authorities, User Services, Policy Authorities, and Users interact and securely share attributes, policies and other HGABAC information. Special care is taken to ensure the architecture will support distributed and federated security domains and that authentication can happen in an *"off-line"* manner (i.e. without having to connect to a third party once an Attribute Certificate has been issued). Additionally, an update to the HGPL policy language is presented (Chapter 5 Section 5.4.5), a namespace for uniquely identifying attributes and other HGABAC elements is introduced (Chapter 5 Section 5.4.1), as well as a cryptographically secure Attribute Certificate document format (Chapter 5 Section 5.4.3) used for sharing user attribute information between services.

Chapter 6: Incorporating Off-Line Attribute Delegation into HGABAC

Chapter 6 builds on the work in Chapters 3 to 5 to create a formal model of User-to-User Attribute Delegation for HGABAC first proposed as one of the strategies in Chapter 4. Extensions to the HGABAC model are proposed to support the new delegation model and a supporting delegation framework (Chapter 4 Section 6.4) is introduced that updates the HGAA architecture and Attribute Certificate for delegation.

Chapter 7: Conclusions and Future Work

Lastly Chapter 7 gives concluding remarks, outlines directions for future research and provides a *"reverse literature search"* detailing the impact this work has had to date in the field of ABAC.

Bibliography

Mohammad A Al-Kahtani and Ravi Sandhu. A model for attribute-based user-role assignment. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 353–362. IEEE, 2002.

Mohammad A Al-Kahtani and Ravi Sandhu. Induced role hierarchies with attribute-based RBAC. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, pages 142–148. ACM, 2003.

D Elliott Bell and Leonard J LaPadula. Secure computer systems: Mathematical foundations. Technical report, DTIC Document, 1973.

John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

Smriti Bhatt. *Attribute-Based Access and Communication Control Models for Cloud and Cloud-Enabled Internet of Things*. PhD thesis, UNIVERSITY OF TEXAS AT SAN ANTONIO, 2018.

CIO Council. Federal identity, credential, and access management (FICAM) roadmap and implementation guidance version 2.0, December 2011. URL https://www.idmanagement.gov/wp-content/uploads/sites/1171/uploads/FICAM_Roadmap_and_Implem_Guid.pdf.

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.

Simon Godik, Anne Anderson, Bill Parducci, Polar Humenn, and Sekhar Vajjhala. OASIS eXtensible access control markup language (XACML). Technical report, OASIS, 2002.

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

Patricia P Griffiths and Bradford W Wade. An authorization mechanism for a relational database system. *ACM Transactions on Database Systems (TODS)*, 1(3):242–255, 1976.

Maanak Gupta, Farhan Patwa, and Ravi Sandhu. An attribute-based access control model for secure big data processing in Hadoop ecosystem. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 13–24. ACM, 2018.

Maanak Gupta, James Benson, Farhan Patwa, and Ravi Sandhu. Dynamic groups and attribute-based access control for next-generation smart cars. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 61–72. ACM, 2019.

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

Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 800:162, 2013.

John Hughes and Eve Maler. Security assertion markup language (SAML) v2. 0 technical overview. Technical report, OASIS, 2005.

Junbeom Hur and Dong Kun Noh. Attribute-based access control with efficient revocation in data outsourcing systems. *Parallel and Distributed Systems, IEEE Transactions on*, 22(7): 1214–1221, 2011.

Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012.

Butler W Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2): 169–180, 2009.

Matunda Nyanchama and Sylvia Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):3–33, 1999.

Office of the Press Secretary. National strategy for information sharing and safeguarding. http://www.whitehouse.gov/the-press-office/2012/12/19/national-strategy-information-sharing-and-safeguarding, December 2012. Accessed: 2014-12-07.

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

Daniel Servos. A role and attribute based encryption approach to privacy and security in cloud based health services. Master's thesis, Lakehead University, 2012.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *The 7th International Symposium on Foundations & Practice of Security FPS'2014*, volume 8930. Lecture Notes in Computer Science (LNCS), 2014.

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017.

Daniel Servos, Sabah Mohammed, Jinan Fiaidhi, and Tai hoon Kim. Extensions to ciphertext-policy attribute-based encryption to support distributed environments. *International Journal of Computer Applications in Technology*, 47(2):215–226, 2013.

Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, pages 735–737. ACM, 2010.

Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

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

Xinwen Zhang, Yingjiu Li, and Divya Nalla. An attribute-based access matrix model. In *Proceedings of the 2005 ACM Symposium on Applied computing*, pages 359–363. ACM, 2005.

Chapter 2

Current Research and Open Problems in Attribute-Based Access Control

Reprint Notice

A version of this chapter with the background martial from Chapter 1 has been published in ACM Computing Surveys as:

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. ***ACM Computing Surveys (CSUR)*****, 49(4):65, 2017**

https://doi.org/10.1145/3007204

© 2017 Association for Computing Machinery, Inc. Reprinted by permission. Copyright permission and license can be found in Appendix A.1.

2.1 Introduction & Methodology

This chapter provides a review of ABAC research as of October 20th 2014 (the date the survey was conducted) with a focus on models that incorporate attribute-based concepts. A novel taxonomy of current ABAC efforts is presented (in Section 2.2) and used to classify and organize each work.

A structured approach was used to locate peer-reviewed literature related to ABAC for the purposes of this literature survey. Searches for refereed journal papers, conference papers and dissertations were conducted using the Google Scholar[1] and DBLP[2] search engines with queries relating to ABAC (e.g. searching for paper titles containing “attribute-based access control”, “ABAC”, “attribute-based”, etc.). Articles were then manually reviewed for inclusion/exclusion based on the following criteria:

[1]http://scholar.google.ca
[2]http://dblp.org/search/

Inclusion Criteria:

Papers and articles discussing models, implementations, frameworks and architectures involving ABAC were included. Works dealing with attribute-based policies and policy languages were also included as well as works describing attribute sharing, storage and privacy but are not discussed in this chapter beyond their inclusion in the taxonomy in Section 2.2 and statistics given in this section.

Exclusion Criteria:

- Any non-refereed work including patents, standards (XACML and SAML are mentioned due to their frequent use in refereed literature, but not included in the statistics in this document), technical reports, or special publications (*The NIST Guide to Attribute Based Access Control (ABAC) Definition and Considerations* [Hu et al. 2013] is discussed in the introduction of this chapter but not included in the statistics).

- Any work that is not in English or is incomprehensible due to language issues (e.g. poorly translated articles).

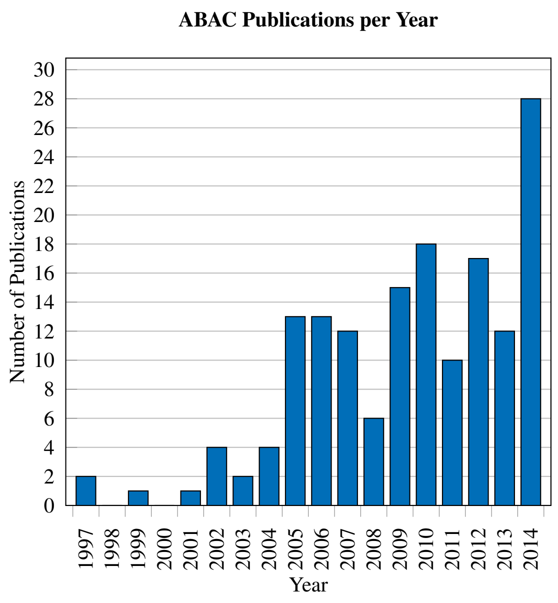
- Only documents published on or before October 20th 2014 are included (due to the date the survey was conducted). No documents were excluded based on age with the earliest document included being published in 1997.

- Any literature related to, or primarily using Attribute-Based Encryption (ABE) was excluded as this literature search is intended to focus on models, frameworks, architectures and use of ABAC as opposed to attribute-base cryptography. While ABE may be a useful tool for enforcing attribute-based policies in environments where traditional policy enforcement is not possible (e.g. in off-line or untrusted environments), it in its self does not provide an underlying model for access control and only comprises one component of a complete security architecture.

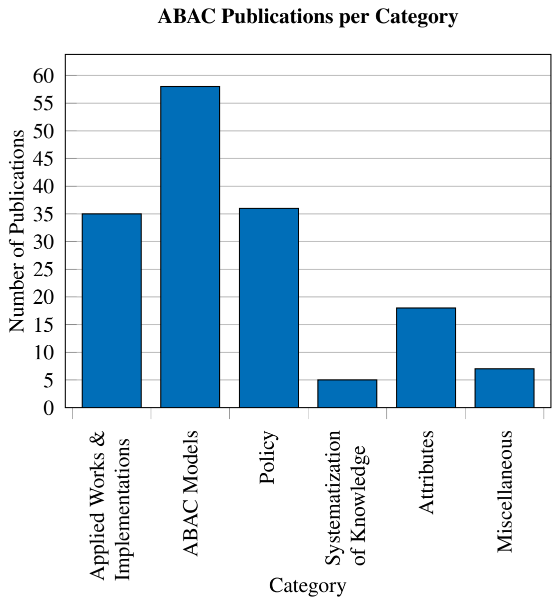
- Any article that was superseded by another work by the same authors is excluded for the newer work. For example, if an author published the beginnings of an ABAC model in a conference and then further developed and finalized this model in a later journal paper, both works are considered to be the same model (both are included in the statistics in Figure 2.1).

The result of this manual search found 199 papers that fall into at least one of the categories described in Section 2.2. A summary of the year in which each paper was published is given in Figure 2.1a and the category to which it belongs in Figure 2.1b. From this set of papers, the most notable and relevant from the category of "ABAC Models" and its child categories are reviewed in Section 2.3 in this chapter.

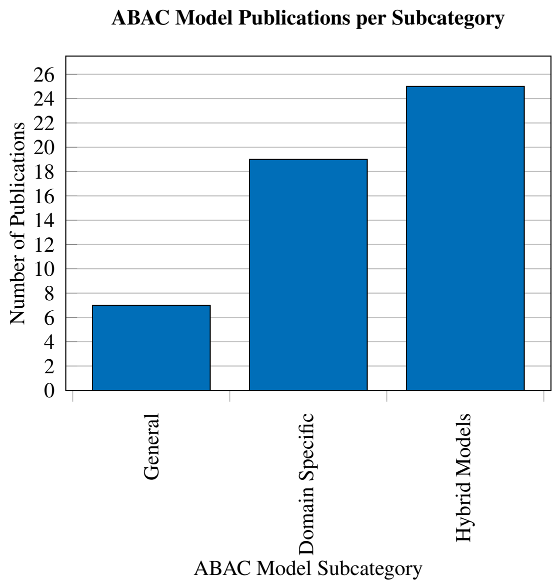
The remainder of this chapter is divided into the following sections: Section 2.2 provides a taxonomy of current areas of ABAC research, Section 2.3 reviews the most notable ABAC models and frameworks, Section 2.4 identifies and discusses open problems not yet addressed by present ABAC efforts, and Section 2.5 provides concluding remarks.



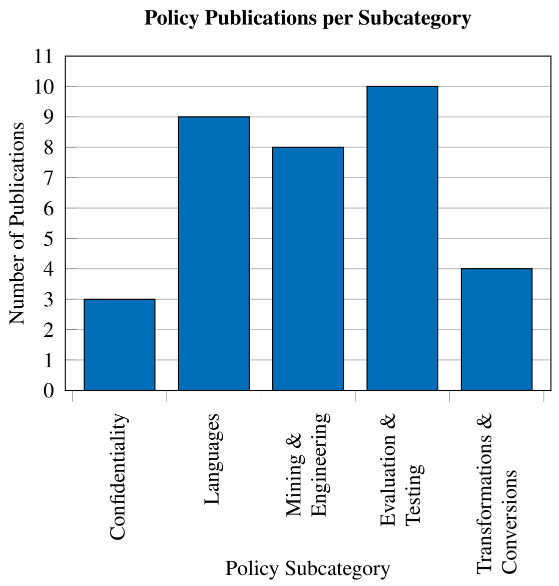
(a) ABAC publications per year between January 1st, 1997 and October 20th, 2014.



(b) ABAC publications per category as defined in Section 2.2.



(c) ABAC model Publications per subcategory as defined in Section 2.2.



(d) Policy publications per subcategory as defined in Section 2.2.

Figure 2.1: Distribution of ABAC related works matching criteria outlined in Section 2.1.

2.2 Taxonomy of Current Areas of Research

The current body of ABAC related research can be classified into a number of hierarchical categories as described in Section 2.2. This taxonomy of current ABAC research was created after manually analyzing the peer-reviewed literature found via the methodology described in Section 2.1 and grouping works describing similar aspects of ABAC together. Related groups (e.g. Policy Languages and Policy Mining) were then further grouped under a more general category (e.g. Policy) that adequately describes all members of the child groups. A diagram of the taxonomy is presented in Figure 2.2.

2.3 Models and Frameworks

2.3.1 Pure ABAC Models

Recent efforts have aimed to take the first steps towards creating foundational models of "pure" ABAC (i.e. ABAC models that are not simply extensions to existing models, e.g. RBAC, but new attribute-based models that can be seen as a generalization of traditional models). A summary of the most relevant attempts at creating such a model are given in Tables 2.3 and 2.4 on pages 25 and 31, with a more in-depth review of each being given later in this section. These efforts can be subdivided into two categories (as described in Section 2.2 and Figure 2.2), "general" and "domain specific". "Domain specific" models aim to provide ABAC for a specific use cases such as cloud computing, web services, etc. while "general" models aim to provide an ABAC solution that may be applied to any situation where access control is desired.

2.3.1.1 General Models

A Logic-Based Framework for Attribute-Based Access Control Wang et al. put forth one of the first "pure" and "general" ABAC models (published in 2004) in the form of a logic-based framework based on logic programming where policies are specified as "stratified constraint flounder-free logic programs that admit primitive recursion" [Wang et al. 2004] and attributes and operations are modelled as sets in computable set theory [Dovier et al. 2000]. Methods of optimizing the runtime performance of evaluating an ABAC-based policy are also demonstrated, which involve transforming a given ABAC policy into a semantically equivalent but runtime and overhead reduced policy when possible. While Wang et al.'s framework introduces hierarchical attributes (something lacking from other models), it is largely focused on the representation, consistency and performance of attribute-based policies and their evaluation. Several critical components are absent, including lacking object attributes (the only attributes considered are user attributes) and omitting formalization of ABAC aspects outside of policies and their evaluation (e.g. only access control on services/operations is considered).

Attribute-Based Access Matrix Model Zhang et al.'s 2005 paper proposes a unique model of ABAC based around an attribute enhanced access matrix, called the Attribute-Based Access Matrix (ABAM) model [Zhang et al. 2005]. ABAM defines an access matrix in which each row is represented by a pair consisting of a subject and its set of attributes $(S_i, ATTS(S_i))$

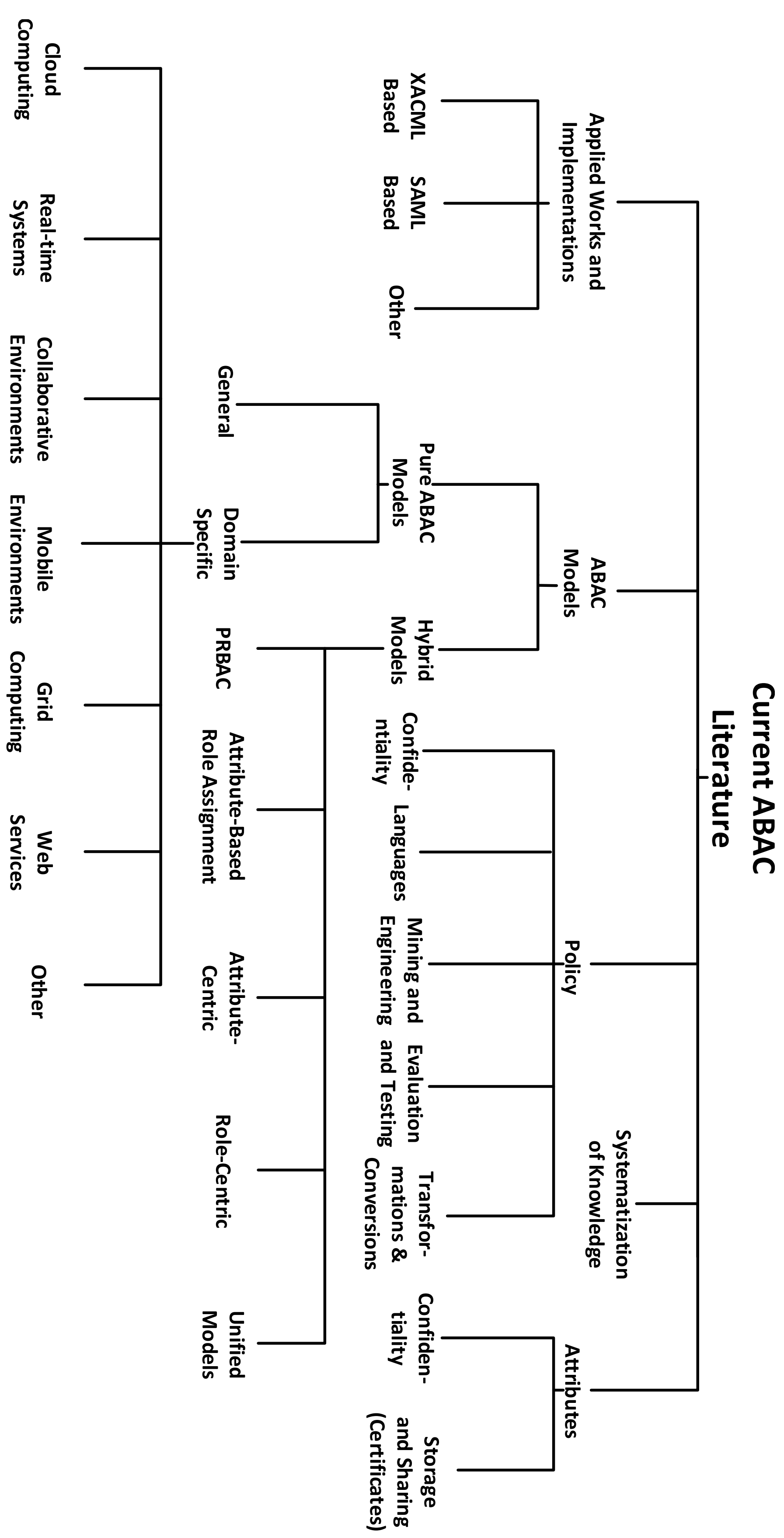


Figure 2.2: Taxonomy of current research areas in ABAC. Each category is described in Section 2.2.

Table 2.1: Description of Taxonomy Categories in Figure 2.2.

Category	Description
Applied Works & Implementations	Literature describing implementations of ABAC systems, frameworks for using XACML, SAML, etc. or any kind of application of existing ABAC research.
• XACML-Based	Implementations or applied work using XACML.
• SAML-Based	Implementations or applied work using SAML.
• Other	Any literature describing implementations or applied work that does not fit into the above subcategories.
ABAC Models	Literature describing access control models that incorporate attributes into access control decisions.
• Hybrid Models	Models that extend or combine existing (non-ABAC) models of access control (e.g. RBAC) to incorporate attributes.
– PRBAC	Parameterized Role-Based Access Control (PRBAC) models are RBAC models based around extending RBAC by parametrizing permissions and/or roles as described in Section 2.3.2.1.
– Attribute-Based Role Assignment	Models that extend RBAC to add attributes as described in Kuhn et al.'s Dynamic Roles strategy (i.e. assigning roles via user attributes). Described in Section 2.3.2.2.
– Attribute-Centric	Models that extend RBAC to add attributes as described in Kuhn et al.'s Attribute-Centric strategy that would not be classified as "pure" models of ABAC. Described in Section 2.3.2.3.
– Role-Centric	Models that extend RBAC to add permission filtering based on attributes as described in Kuhn et al.'s Role-Centric strategy. Described in Section 2.3.2.4.
– Unified Models	Access control models that combine ABAC with with alternative access control models (i.e. non-traditional models) as described in Section 2.3.2.5.
• Pure ABAC Models	ABAC models that are not extensions to existing models of access control but new attribute-based models.
– General	ABAC models that are system independent in that they are general enough to be applied to any access control use.
– Domain Specific	ABAC models that are designed for a particular domain or use (e.g. for protecting web services).
*** Cloud Computing**	Models targeting the domain of cloud computing.
*** Real-time Systems**	Models targeting the domain of real-time systems.
*** Collaborative Environments**	Models targeting the domain of collaborative work and educational environments.
*** Mobile Environments**	Models targeting the domain of mobile environments, including both systems that track mobile physical objects and mobile computing systems (e.g. cell phones).
*** Grid Computing**	Models targeting the domain of grid computing.
*** Web Services**	Models targeting the domain of web services, including service oriented architectures.
*** Other**	Any domain specific model that does not fit in one of the above child categories.
Policy	Literature describing the mining for or evaluation, testing, and development of attribute-based policies and languages. Also includes works attempting to preserve the privacy of policies or otherwise hide details of policies from an adversary.
• Confidentiality	Works aimed at preserving the privacy of attribute-based policies or otherwise hide details of policies from an adversary.
• Languages	Literature describing or extending attribute-based policy languages.
• Mining & Engineering	Research aimed at the automatic mining of attribute-based policies or otherwise engineering attribute-based policies.
• Evaluation & Testing	Literature describing the testing and evaluation of attribute-based policies. Includes both the implementation of tools to automate the testing of policies and efforts to prove the security/safety of policies (formally or otherwise).
• Transformations & Conversions	Methods and techniques for converting access control policies between policy languages, access control models, or otherwise transforming existing policies (e.g. to optimize policy evaluation or analysis).
Systematization of Knowledge	Literature reviews and systematization of knowledge in the area of ABAC.
Attributes	Works relating to sharing, storing, validating, securing or ensuring the privacy of attributes used in ABAC.
• Confidentiality	Efforts to ensure the privacy of attributes. That is protecting unwanted entities from determining the value of potentially sensitive attributes.
• Storage & Sharing (Certificates)	Efforts to enable the sharing or storage of attributes. Includes frameworks, protocols and data structures (e.g. attribute certificates) for securely sharing attributes between access control entities.

Table 2.2: Column Legend for Tables 2.3 and 2.4

Column	Description
Object Attributes	Whether the model supports object or resource attributes.
User Attributes	Whether the model supports user or subject attributes.
Environment Attributes	Whether the model supports attributes that describe the systems environment (e.g. current time, number of online users, etc.).
Connection Attributes	If the model supports attributes relating to the subject's session and/or connection to the system (e.g. subject's host name, IP, session ID, etc.).
Mutable Attributes	If the model supports attributes whose value change as a result of a subject's requests on a system.
Policy Language	If the model formalizes its own policy language (✓) or the language being used (e.g. XACML).
Hierarchical	Whether the model supports hierarchical constructs to simplify administration and/or increase flexibility of policies. (e.g. hierarchical attributes, hierarchical user or object groups, etc.).
Recursive Rules	If the policy language presented in the work supports recursive rules or policies.
Trust	Whether the model incorporates the notion of trust similar to that found in trust-based access control.
User & Object Groups	Whether the model supports user or object groups to simplify administration and/or increase flexibility of policies.
Separation of Duties	Whether the model supports any kind of separation of duties and the types supported (e.g. static, dynamic, etc.).
Delegation	If subjects are able to delegate a subset of their attributes or privileges to other subjects.
Functional Specification	Whether a functional specification is provided with the model.
Formal Model	If the model is formalized (i.e. if any formal language or notation is used to fully describe the model).
Emulates Traditional Models	If it is shown that the model can emulate the traditional models of access control (e.g. DAC, MAC, RBAC).
Administration Model	If an administrative model or functions are defined or presented.
Full Model	Whether all necessary components of an usable ABAC model are described in the cited work. Necessary components include; definition and format of attributes, a description of policy language used or how polices are stated/evaluated, and details of relations between access control elements.
Extends	The models extend or used as the basis to create this hybrid ABAC model.
Identityless	If this hybrid ABAC model allows for identityless access control. That is, access control that does not require pre-existing knowledge about the user or their roles in the system.

Table 2.3: Comparison of General ABAC Models.
An explanation of each column item can be found in Table 2.2.

	[Wang et al. 2004]	[Jin et al. 2012a]	[Zhang et al. 2005]	[Rubio-Medrano et al. 2013]	[Servos and Osborn 2014]	[Ferraiolo et al. 2011]
Object Attributes	✗	✓	✓	✓	✓	✓(Attributes do not have values)
User Attributes	✓	✓	✓	✓	✓	✓(Attributes do not have values)
Environment Attributes	✗	✗	✗	✓	✓	✗
Connection Attributes	✗	✗	✗	✗	✓	✗
Mutable Attributes	✗	✗	✗	✗	✗	✗
Policy Language	Has method of representing policies but no defined language	✓	No details given for how policies are represented	No policy language use (left to future work)	✓	Policies expressed as chain of attribute assignments
Hierarchical	Hierarchical attributes	✗	✗	✗	Hierarchical user and object groups	Hierarchical attributes
Recursive Rules	✓	✗	✗	Supported via cycles in the TP-Graph	✗	✗
Trust	✗	✗	✗	✗	✗	✗
User & Object Groups	✗	✗	✗	✗	✓	✗
Separation of Duties	✗	✗	✗	✗	✗	✓
Delegation	✗	✗	✗	✗	✗	✗
Functional Specification	✗	✓	✗	✗	✗	✗
Formal Model	✓	✓	✓	Largely informal	✓	✓
Emulates Traditional Models	Not demonstrated	✓	Not demonstrated	Not demonstrated	✓	✓
Administration Model	✗	Limited	Very limited	✗	✗	✓
Full Model	Only models policies and their evaluation	✓	✓	✓	✓	✓

and each column by a pair consisting of an object and its set of attributes (O_i, $ATTS(O_i)$). Each cell ($[S_i, O_i]$) then corresponds to the set of access rights the subject (S_i) may exercise over the object (O_i) assuming certain policies are fulfilled. Operations (or "commands" as they are called in ABAM) may be executed by a given subject over a given object only if the matching access rights required by the operation are found in the access matrix and the subject and object's attributes fulfill the set of policies on the operation.

In addition to the formalization of ABAM, Zhang et al. also provide a safety analysis to prove the decidability of ABAM for a case where the set of attributes is finite, and the attribute relationships allow no cycles. While ABAM's unique use of an access matrix allows for a more auditable ABAC system than other models (basic checks on which users may access a certain object may be accomplished with a simple matrix lookup rather than evaluating policies on a large set of attributes and subjects), it omits details on how policies are administered, composed, or evaluated. A policy language is shown in examples but never formalized fully. Similarly, it is stated that ABAM is comprehensive enough to encompass the traditional access control models; however, this is not demonstrated and it is left unclear how ABAM might encompass MAC or hierarchical RBAC. Lastly, ABAM lacks connection, environment and hierarchical attributes as well as constraints to enforce separation of duty or enable delegation.

Secure Collaborations with Attribute-Based Access Control A more recent work (2013) by Rubio-Medrano et al. [Rubio-Medrano et al. 2013] introduces the notion of security tokens into an abstract model of ABAC that defines the relevant core components and attributes required for a minimal reference model. Unlike other rule-based ABAC models that make access control decisions on the basis of evaluating policies given the current state of various attributes, Rubio-Medrano et al.'s model maps attributes of access control entities (subjects, objects, etc.) to security tokens by traversing an administrator defined Token-Provisioning Graph (TP-Graph). The TP-Graph is a directed, possibly cyclic, graph whose vertices represent sets of related attributes or security tokens (referred to as attribute or security token families) and its edges represent Token-Provisioning Functions (TP-Functions) that map attribute or security token families to a different security token family based on defined criteria the attribute or token values must meet. By allowing system administrators to define TP-Functions and relating security tokens to the permissions (object, operation pairs) they grant, it enables access control decisions that are claimed to be more auditable and open to security analysis using techniques based on graph theory.

While Rubio-Medrano et al.'s model gives a novel take on ABAC, the added auditability and graph-based security analysis come at the cost of increased administrative complexity and overhead. In theory the TP-Graph should allow for the development of security analysis techniques based on graph theory but this seems to be largely left to future works. Additionally, the ABAC model itself is largely informal, leaving most concepts well described but not defined formally. It is left unclear how TP-Functions and the TP-Graph may be created by an administrator or in what form they may take (a policy language is hinted at when directions for future work are discussed). Similarly, no precise description or algorithm is given for how the TP-Graph is traversed or how cycles may be handled.

ABAC$_\alpha$ Another recent (2012) work by Jin et al. aims "to develop a formal ABAC model that is just sufficiently expressive to capture DAC, MAC and RBAC" [Jin et al. 2012a]. This model, $ABAC_\alpha$, provides formalizations of the basic ABAC elements (users, objects, policies, etc.), their relations and constraints that allow emulation of the traditional models. A partial policy and constraint language, called Common Policy Language (CPL), based on set theory notation and Boolean logic is defined and example configurations are given for DAC, MAC, and RBAC-style access control in $ABAC_\alpha$. Additionally, a limited functional specification including a bare minimum of administrative functions is specified (although details on what authorization conditions may be required for administrative functions are not given).

CPL is used for both policy specification and configuring constraints on $ABAC_\alpha$ to limit possible attribute assignments and set a valid range and type of attribute values. Example 2.1 shows an authorization policy in CPL for enforcing RBAC style access control. In this case S is the set of all subjects, O is the set of all objects, *srole* is a subject attribute that contains the subjects roles, *rrole* is an object attribute that contains the set of roles that grant permission to read the object and *wrole* is an object attribute that contains the set of roles that grant permission to write to the object. The authorization policy states that a subject can only read the object if they have a role in the objects *rrole* attribute value set and can only write to the object if they have a role in the objects *wrole* attribute value set.

Example 2.1. Simple (non-hierarchical) RBAC authorization policy:

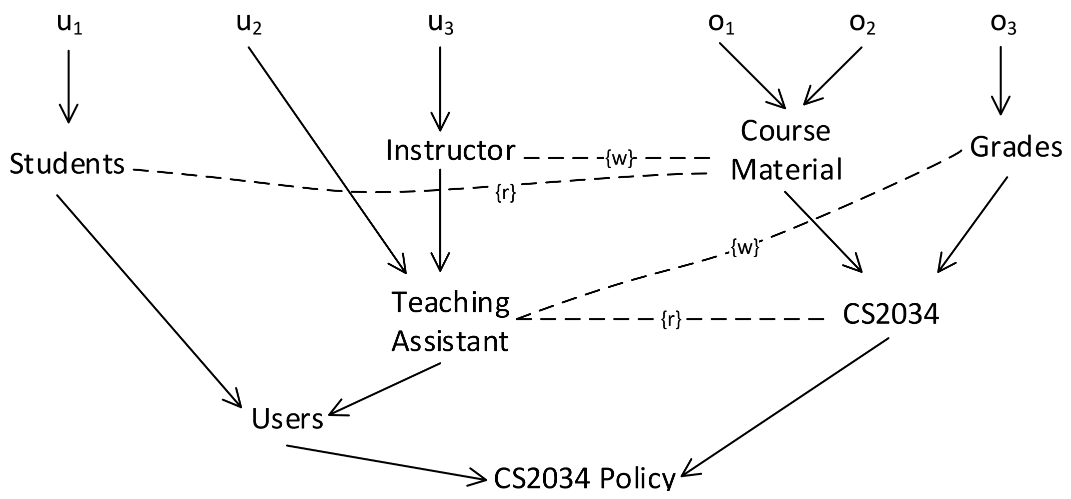
$$\texttt{Authorization}_{\texttt{read}}\texttt{(s : S, o : O)} \equiv \exists \texttt{r} \in \texttt{srole(s)} \in \texttt{rrole(o)}$$
$$\texttt{Authorization}_{\texttt{write}}\texttt{(s : S, o : O)} \equiv \exists \texttt{r} \in \texttt{srole(s)} \in \texttt{wrole(o)}$$

While this work provides a sufficient basis on which new foundational models of ABAC may feasibly be built, it (intentionally) lacks several necessary components for the real world. Features such as attribute and object hierarchies, environment and connection attributes, delegation and separation of duties are omitted and left to future models built upon $ABAC_\alpha$. Finally, the given policy language, while adequate for modelling traditional access control systems, is insufficient for real world application. No specifics are given on how CPL might handle multiple policy composition or conflicting policies and the heavy use of set theory notation in the language (as opposed to traditional Boolean statements) makes CPL's practicality over an existing policy language such as XACML questionable (creating XACML profiles for $ABAC_\alpha$ is left to future works).

The Policy Machine Ferraiolo et al. have developed a novel approach to access control that is highly attribute-based in the form of the Policy Machine (PM) [Ferraiolo et al. 2011, 2015]. The PM is an architecture and access control framework to support the specification and enforcement of attribute-based access control policies that aims to redefine and generalize access control to provide a unified mechanism under which a wide range of policies may be enforced. Unlike other approaches that define attributes as name value pairs, the PM represents user attributes as many-to-many relations between users and capabilities (operation object pairs that grant the ability to perform the given operation on the given object). Similarly, object attributes are defined as many-to-many relations between sets of objects and sets of access entries (user operation pairs that state that the given user may perform the given operation). Attributes are hierarchical, allowing attributes to be assigned to other attributes so long as the chain of assignments remains acyclic. If two user attributes ua_1 and ua_2 exist such that ua_1 is assigned to ua_2, the set of users assigned to ua_1 are contained in ua_2 and the capabilities granted from ua_1 are those obtained through the chain of attribute assignments (e.g. all users assigned to ua_1 in this case would gain the capabilities granted from both ua_1 and ua_2). Assignments between object attributes work in a similar manner. If two object attributes oa_1 and oa_2 exist such that oa_1 is assigned to oa_2, the set of objects assigned to oa_1 are contained in oa_2 and the objects of oa_1 have the access entries assigned to oa_2 (in addition to those assigned to oa_1).

Policies are specified using policy classes, chains of attribute assignments terminating with a policy class as shown in the example policy given in Example 2.2. In this example, an RBAC style policy class is shown that governs access to materials and grades for a university course. The user attribute *Instructor* grants the capability to write to objects assigned the *Course Material* attribute (in this case o_1 and o_2), however, as the *Instructor* attribute is assigned the *Teaching Assistant* attribute it also grants the capabilities of the *Teaching Assistant* attribute (and all other user attributes on the path to the policy class in Example 2.2). The *Teaching Assistant* attribute grants the capability to read all objects assigned with the *CS2034* attribute. This includes any objects assigned attributes that are in turn assigned the *CS2034* attribute (i.e. the *Course Material* and *Grades* object attributes) in the assignment chain. In this example the resulting permissions allow teaching assistants (i.e. u_2) to read all of the CS2034 objects (o_1, o_2, and o_3) but only write to the grade objects (o_3). Instructors (i.e. u_3) have all permissions of teaching assistants in addition to being able to write to Course Material objects (o_1 and o_2). Finally, Students (i.e. u_1) are limited to only reading Course Material objects (o_1 and o_2).

Example 2.2. Example Policy Machine policy class. Solid arrows represent attribute assignments, while dashed lines represent capabilities of the shown user attributes.



Ferraiolo et al. show that the PM is sufficiently flexible to enforce DAC, MAC, RBAC and Chinese Wall [Brewer and Nash 1989] style security policies and provide further means to constrain policies with prohibitions, restrictions and obligations. An administration model is also presented, as well as details on a number of architectural components necessary for implementation. The PM specification described in [Ferraiolo et al. 2015] has served as the basis for the ANSI/INCITS Next Generation Access Control standardization effort [INCITS 2013, 2015].

Hierarchical Group and Attribute-Based Access Control Lastly, the work by Servos and Osborn [Servos and Osborn 2014] (part of this thesis and discussed in-depth in Chapter 3) attempts to create a formal general model of ABAC that provides a group based hierarchical representation of object and user attributes. In this model, entitled Hierarchical Group and Attribute-Based Access Control (HGABAC), attributes are assigned both directly to access control entities and indirectly assigned through user and object attribute groups. Attribute groups help simplify administration of ABAC systems by allowing administrators to create user or object groups whose membership indirectly assigns sets of attribute/value pairs to its members. These groups are hierarchical and inherit attribute/value pairs from their parent groups allowing for more flexible policy representation when combined with the three-valued logic based policy language proposed in the work.

The HGABAC policy language (HGPL) represents policies as C style boolean statements that may evaluate to *TRUE*, *FALSE* or *UNDEFINED*. A resulting evaluation of *TRUE* implies that access should be granted, *FALSE* that it should be denied and *UNDEFINED* if the policy can not be properly evaluated at the current time (equivalent to a result of *FALSE* for access control decision purposes). Policies are associated with a set of operations that they grant if satisfied. Example 2.3 presents a number of example policies that are possible in HGABAC.

Example 2.3. Possible policies supported by HGABAC:

- P_1 `= (user.age >= 18 AND object.title = "Adult Only Book", read)`
  Any user with an age of 18 or older can read the book with the title ”Adult Only Book”.

- P_2 `= (user.id = object.author, write)`
  A user can write to any object they are an author of.

- P_3 `= (user.role IN {"doctor", "intern", "staff"} AND user.id != object.patient, read)`
  A user can read a medical record if they have the role of doctor, intern or staff but only if they are not listed as a patient in that record.

- P_4 `= (object.type = "program" AND object.required_certifications SUBSET user.certifications, run)`
  A user can run a program if they have the required certifications listed in the programs required_certifications attribute.

Servos and Osborn show that their policy language and attribute groups are capable of emulating MAC, DAC and hierarchical RBAC (though not separation of duties) and that their attribute groups result in less complex (in terms of the number of assignments and relations between access control entities) representations than standard (non-hierarchical) ABAC models under a number of hypothetical use cases.

2.3.1.2 Domain Specific Models

While a handful of recent ABAC related works have sought to create “general” models, the more popular trend in modern access control literature has been the creation and formalization of “domain specific” ABAC models (summarized in Table 2.4). A large focus has been given to the domains of cloud computing [Buehrer and Wang 2012], grid computing [Lang et al. 2009, 2006, 2010], web services [Yuan and Tong 2005; Shen and Hong 2006; Xia and Liu 2009; Shen 2009], and related areas including mobile computing [Covington and Sastry 2006] and cross-domain service-oriented architecture [Dan et al. 2012].

Cloud Computing Buehrer and Wang propose an ABAC model based on class algebra, entitled Class Algebra Attribute-Based Access Control (CA-ABAC), intended to provide access control between federated educational clouds [Buehrer and Wang 2012]. CA-ABAC makes use of the non-probabilistic version of class algebra implemented by the Cadabia knowledge base [Buehrer et al. 2001] as a basis for its ABAC policy language. Example policies from [Buehrer and Wang 2012] are given in Examples 2.4 and 2.5.

Example 2.4. Only students that have signed the contract/consent form (the form named okJim55) may read or execute course material owned by the teacher named Jim.

```
new Policy[n1] {
  agents:  "ENV.user in @School[A;B]
    .student{@Form[okJim55] in singedForms}" ∧∧Query
  rights:  @Rights[read,execute],
  objects:  "@Thing{owner = @Teacher[Jim]}
}
```

Example 2.5. Uses the environments time attribute to block students from reading the answers to homework assignment 5 until after the due date.

```
new BlockPolicy[n5] {
  agents:  "ENV.user in @Student," ∧∧Query
  rights:  @Rights[read],
  objects:  "@Homework[assignment5]
    {dueDate > ENV.date}.answer" ∧∧Query
}
```

Buehrer and Wang outline a very informal description of their model which mostly describes policy use and a hypothetical system architecture. While the prospect of using class algebra as a policy language may have potential, CA-ABAC lacks formalization or details on many of the key features of an ABAC system. No description is given of what constitutes an attribute in the model or their relation to users, objects or the environment. Basing the policy language on Cadabia queries may lead to problems for real world use as the Cadabia open source project is no longer maintained.

Real-time Systems Burmester et al. [Burmester et al. 2013] put forward the real-Time Attribute-Based Access Control (T-ABAC) model, that adds real-time attributes to the concept of rule-based ABAC to support highly dynamic real-time applications. Real-time attributes are defined as attributes whose value depends on time and is a member of an ordered set of availability labels which determines the “priority” of a subject’s request, the “congestion” of a resource or the “criticality” of the environment. Burmester et al. also provide a packet forwarding protocol that takes the priority of access requests into account and demonstrate the

Table 2.4: Comparison of Domain Specific ABAC Models
An explanation of each column item can be found in Table 2.2.

	[Buehrer and Wang 2012]	[Burmester et al. 2013]	[Smari et al. 2009, 2014]	[Liang et al. 2012]	[Covington and Sastry 2006]	[Kerschbaum 2010]	[Lang et al. 2006, 2009]
Domain	Cloud Computing	Real-time Systems	Collaborative Environments	Collaborative Environments	Mobile Environments	Mobile Environments	Grid computing
Object Attributes	✓	✓	✓	✓	✓	✓	✓
User Attributes	✓	✓	✓	✓	✓	✓	✓
Environment Attributes	✓	✓	✗	✓	✓	✗	✓
Connection Attributes	✗	✗	✗	✗	✗	✗	Shown in example but not model
Mutable Attributes	✗	✗	Mutable trust attribute	✗	Limited, based on transaction attributes	✗	✗
Policy Language	Class Algebra (from Cadabia knowledge base)	Does not mention policies	Policy language shown in examples but not defined	XACML	Claims to have policy language but is left undefined and no examples given	XACML	Policies are algorithms, no language used/defined
Hierarchical	✗	✗	✗	✗	✗	✗	✗
Recursive Rules	✗	✗	✗	✗	✗	✗	✗
Trust	✗	✗	✓	✗	✗	✗	✗
User & Object Groups	✗	✗	✗	✗	✗	✗	✗
Separation of Duties	✗	✗	✗	✗	✗	✗	✗
Delegation	✗	✗	✗	✗	✗	✗	✗
Functional Specification	✗	✗	✗	✗	✗	✗	✗
Formal Model	Informal	Only formalizes real-time attributes and packet mechanics	✓	✓	Informal	✓	✓
Emulates Traditional Models	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated
Administration Model	✗	✗	✗	✗	✗	✗	✗
Full Model	Lacks details, mostly describes policy use	Only models real-time attributes and packet mechanics	Lacks details, unclear how policies are evaluated and format of attributes	Lacks details, more architecture then model	Lacks details, policy language not formalized	✓	✓

Continued on next page in Table 2.5.

Table 2.5: Comparison of Domain Specific ABAC Models (continued from Table 2.4). An explanation of each column item can be found in Table 2.2.

	[Lang et al. 2010]	[Yuan and Tong 2005]	[Shen and Hong 2006]	[Dan et al. 2012]	[Xia and Liu 2009]	[Shen 2009]	[Zhang et al. 2014]
Domain	Grid computing	Web Services	Web Services	Web Services	Web Services	Web Services	Web Services
Object Attributes	✓	✓	✓	✓	✓	✗	✓
User Attributes	✓	✓	✓	✓	✓	✓	✓
Environment Attributes	✓	✓	✓	✓	✓	✗	✓
Connection Attributes	✗	✗	✗	✗	✗	✗	✗
Mutable Attributes	✗	✗	✗	✗	✗	✗	✗
Policy Language	XACML	Model lacks language, implementation uses XACML	XACML	Model lacks language, implementation uses XACML	XACML	XACML	XACML
Hierarchical	✗	✗	✗	✗	✗	✗	✗
Recursive Rules	✗	✗	✗	✗	✗	✗	✗
Trust	✗	✗	✗	✗	✗	✗	Claims trust attribute but fails to provide details
User & Object Groups	✗	✗	✗	✗	✗	✗	✗
Separation of Duties	✗	✗	✗	✗	✗	✗	✗
Delegation	✗	✗	✗	✗	✗	✗	✗
Functional Specification	✗	✗	✗	✗	✗	✗	✗
Formal Model	Largely informal	Simplistic	Simplistic	Simplistic	✓	Informal	Largely Informal
Emulates Traditional Models	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated	Not demonstrated
Administration Model	✗	✗	✗	✗	✗	✗	✗
Full Model	Minimal model, mostly architecture combining existing works	✓	✓	More implementation using XACML then model	✓	More theoretical architecture combining existing works then model	Basic definitions for model, mostly architecture combining existing works.

versatility of T-ABAC by discussing two possible applications, a substation automation system, and a medical CPS. While the T-ABAC model does a good job of dealing with issues unique to real-time systems, it omits several core ABAC model components. No information is given about how policies are represented, evaluated or apply to the model and only the concept of real-time attributes is developed with regard to ABAC. As such, T-ABAC presents a sufficient basis for extending existing ABAC models to support real-time applications, but is missing necessary components to be a standalone model.

Collaborative Environments Collaborative working and educational environments enable cooperative work, research and learning through shared application or service resources. Collaborative applications and services include but are not limited to E-mail, wikis, instant messaging, group blogs, version control systems, courseware, and software to support shared document, workspace, task and work flow management. As these applications have unique access control requirements, they have attracted a notable amount of attention in the access control literature including a number of papers focusing on applying ABAC policies to collaborative systems. Such works include Smari et al.'s ongoing research project and multiple publications supporting ABAC for collaboration environments [Zhu and Smari 2008; Smari et al. 2009, 2014] and Liang et al.'s multiple-policy supported ABAC architecture for large-scale collaboration systems (MPABAC) [Liang et al. 2012].

Smari et al. present an ABAC model aimed at collaboration environments [Zhu and Smari 2008] that incorporates trust and privacy into access control policies. They extend this model over a number of works [Smari et al. 2009, 2014] to fully formalize their notion of trust and privacy and illustrate their model with an implementation and detailed case study involving a multi-organizational collaborative crisis management system. Their model consists of a three-valued ("allow", "deny", and "NA") rule-based policy evaluation on subject and object attributes that integrates trust and privacy through special mutable trust and purpose attributes. Trust is considered to be "the degree that a subject will perform as expected in a certain given context" and is quantified as a real number between 0 and 1 and assigned as the value of a subject's trust level attribute. As a user performs requests upon the system, their previous behaviour is assessed and used to determine if their future behaviour deviates or conforms to what is expected (effecting the user's trust level). In addition to this dynamic notion of trust, a subject's trust level is also dependent on other subject attributes including the recommendation from others and the level of collaboration between organization of a requester and that of a resource. This trust level can then be included in access control policies to limit or expand a user's access to system resources based not only on traditional access policies but also their evaluated trust level. The concept of privacy is enforced by assigning a set of well-defined purposes to subjects and objects as an attribute which represents either for what purposes a subject may access an object or for what purposes an object may be accessed respectively. Access to a specific object is allowed only if the purpose of the subject for accessing the object matches a purpose allowed by the object. While Smari et al.'s model successfully introduces trust and privacy to ABAC, it omits details on policy evaluation or a formalized policy language. Example policies are shown but no explanation is given for how the operations may work with the three-valued logic used by the model.

Liang et al. offer a model and architecture for Multiple-Policy Attribute-Based Access Control (MPABAC) [Liang et al. 2012] that addresses the access control issues inherent in large-scale device collaboration systems (i.e. mainly the large number of heterogeneous devices). Unlike other ABAC models, MPABAC models resources as devices (device attributes rather than object attributes, etc.) and focuses on limiting access to networked devices (e.g. seismographs, orchestrated lights, etc.) based on multiple policies possibly originating from different domains but evaluated locally. The described architecture and implementation detail how XACML may be used to communicate access control information between different domains and enforce the MPABAC model. As MPABAC largely focuses on architecture and XACML use, the ABAC model itself omits details on how policies are evaluated or combined. Details on how attributes are represented (e.g. if they are sets, collections of values, or primitive data types, etc.) are similarly omitted and the notion of policies having a priority level is introduced but not fully formalized in terms of the MPABAC model.

Mobile Environments Several efforts have advocated models of ABAC that are contextually aware of a user or resource's physical environment. Covington and Sastry's Contextual Attribute-Based Access Control (CABAC) [Covington and Sastry 2006] investigates using the dynamic properties commonly available in a mobile environment (e.g. a user's current physical location) as attributes to support ABAC for mobile applications. Transaction attributes that are mutated or created based on a user's transactions with a service provider (e.g. a user may have an attribute that holds the total amount of money spent at a certain shop) are also supported as a special case of contextual user attribute. These attributes allow for access policies to be based around past transactions with a user. For example, a restaurant may have a policy that grants access to their Wi-Fi connection to customers that have made a purchase in the last 24 hours. A custom authorization policy specification language consisting of constant symbols (e.g. object references), variable symbols (e.g. location and time), and operation symbols (e.g. +, -, /, *, *AND*, *OR*, <, >, etc.) is described but not formalized or demonstrated.

A similar work by Kerschbaum details an access control model for mobile physical objects [Kerschbaum 2010] that aims to apply access control to physical mobile resources embedded with RFID tags. Kerschbaum's model applies attribute-based visibility policies to supply chain information based on the contextual location of physical objects as they transverse multi-company supply chains. This is accomplished by extending Yuan and Tong's ABAC model for web services [Yuan and Tong 2005] (discussed later in this section) to include upstream and downstream visibility as an attribute for each pairing of subject and object to allow policies to be created based on an object's trajectory relative to a subject (i.e. whether a subject is upstream or downstream of an object's current location in the supply chain). Policy rules are specified using a Boolean function of the subject and resources attributes as shown in Example 2.6. In this example a subject, *s*, may access the information pertaining to a resource, *r*, if the attributes "downstream" or "upstream" are in the attribute set produced by the pairing of *s* and *r*, i.e. *ATTR(s, r)*. Such attribute sets are continuously updated based on the subject and resource's current physical location.

Example 2.6. Resource visibility policy:
```
access(s, r) ← "downstream" ∈ ATTR(s, r) ∨
                "upstream" ∈ ATTR(s, r)
```

A method for encoding such visibility policies in XACML is also described. XACML environment attributes are used in place of assigning attributes to pairings of subjects and resources (as XACML does not support direct assignment of attributes to subject resource pairs).

Grid Computing Grid computing has been another common target of domain specific ABAC models as it presents unique access control requirements stemming from the distributed nature of grid computing, where resource providers and users may be in independent security domains. Lang et al.'s Attribute-Based Multipolicy Access Control (ABMAC) [Lang et al. 2006, 2009] presents a model and Globus Toolkit release 4 (GT4) based authorization framework for applying ABAC to grid computing. In addition to user, object and environment attributes, ABMAC supports service and action attributes that allow attributes to be applied to grid services or a grid action respectively. Policies differ from most rule-based ABAC models in that each policy is encapsulated and uses its own definitions and decision-making algorithms, allowing for independent evaluation without changing a policy's description. A similar but more informal work, Grid_ABAC [Lang et al. 2010], also uses GT4 to implement and demonstrate a grid based ABAC model that supports action attributes and uses XACML as a policy language. Grid_ABAC, unlike ABMAC, largely focuses on being a grid authorization architecture and as such provides a more minimalistic ABAC model.

Web Services By far the largest area of research in domain specific ABAC models is towards attribute and policy-based access control for web services. Identity-less access control such as ABAC provides a potential solution to furthering automated web service discovery and use by allowing access control decisions to be made without prior knowledge of the subject or their relation to the service provider. Of the many ABAC models targeting web services [Yuan and Tong 2005; Shen and Hong 2006; Dan et al. 2012; Xia and Liu 2009; Shen 2009; Zhang et al. 2014], most notable is the model by Yuan and Tong (ABAC for Web Services), upon which several other ABAC models [Kerschbaum 2010; Xia and Liu 2009] are based. Yuan and Tong describe ABAC in terms of authorization architecture and policy engineering and give an informal comparison between ABAC and traditional role-based models. Policy rules are defined as a Boolean function comparing the attributes of the subject making the request, the resource potentially being access and the system's environment. If the function evaluates as true, access is granted to the subject, otherwise access is denied.

Two example policy rules from [Yuan and Tong 2005] are shown in Example 2.7. Rule 1 (R_1) allows a subject, *s*, to access the ApprovePurchase web service resource, *r*, if they have a Role attribute with a value of "Manager". Rule 2 (R_2) allows any user access to a resource they own. That is, if their user ID is equal to the value of the ResourceOwner attribute for the given resource, *r*.

Example 2.7.

```
R1:  can_access(s, r, e) ←
       (Role(s) = "Manager") ∧
       (Name(r) = "ApprovePurchase")
R2:  can_access(s, r, e) ←
       (UserID(s) = ResourceOwner(r))
```

While Yuan and Tong's model is limited, only giving an overview of subject, object, and environment attributes and their relation to policies, it was an earlier effort which served as the basis for more formalized future works. In addition to the model, an authorization architecture is introduced that uses XACML to securely communicate attributes, policies, and access control decisions between a number of actors.

Shen and Hong propose WS-ABAC [Shen and Hong 2006], a more extensive but still relatively simplistic ABAC model designed for web services accompanied by an XACML-based authorization architecture. In the WS-ABAC model policies are based on a straightforward tuple language that is mapped to XACML when used in their authorization architecture. Attribute constraints are expressed as a series of attribute conditions, *<Attribtue Name> <Operation> <Value>* statements, combined with logical *AND* (represented as $\cap$) or *OR* (represented as $\cup$) operators. Valid attribute condition operations are limited to $>$, $<$, $\geq$, $\leq$, $=$, $\neq$. In Example 2.8, constraint C_1 limits access to a web service to a manager who is accessing the service between the hours of 9:00 AM and 5:00 PM from the office. Constraint C_2 limits access to clerk when the system load is low or to a manager at any time or system load.

Example 2.8. Example WS-ABAC Attribute Constraints

```
C1:  Identity="manager" ∩ Time≥9:00 ∩ Time≤17:00 ∩ Location="office"
C2:  Identity="clerk" ∩ System_load="low" ∪ Identity="manager"
```

WS-ABAC policies are defined as the triple *<S, srv, C>*, where *S* is the set of subjects to which the policy pertains, *srv* is the service the policy grants access to and *C* is the attribute constraint. Access to a service is only granted if (1) there exists a policy triple containing the requested service, (2) the user, U, making the request is a member of S ($U \in S$) and (3) the attribute constraint, C, evaluates to true. As with Yuan and Tong's ABAC for Web Services, this work presents a minimalistic model and mostly focuses on an architecture that uses XACML and attribute-based policies to provide authentication for web services (as opposed to a complete and/or foundational model of ABAC).

A number of later publications have followed in the same suit, providing minimally sufficient models with accompanying XACML-based architectures targeting web services or Service-Oriented Architectures (SOA). Dan et al. [Dan et al. 2012] create and implement an XACML architecture for cross-domain SOAs. Xia and Liu [Xia and Liu 2009] study using action and attribute-based models for web services and develop a limited ABAC model and XACML architecture that extends the work of Yuan and Tong. Shen [Shen 2009] presents SABAC, an informal semantic-aware ABAC model for web services that makes use of present standards, including XACML. Finally, Zhang et al. [Zhang et al. 2014] describe a largely informal ABAC security model for service-oriented computing that adds the notion of trust as well as offering an authorization architecture for web services based on combining existing works (mainly SAML and XACML). However, few details are provided on their ABAC or trust model, as more attention is given to the authorization architecture.

Digital Libraries An earlier work (2002) by Adam, et al. [Adam et al. 2002] identified the need for attributes to deal with the challenging requirements of providing access control for digital libraries. Digital libraries are information systems that facilitate the storage, retrieval and acquisition of knowledge between creators, consumers and librarians on a global scale.

Adam, et al. suggest a novel access control system for protecting the Global Legal Information Network (GLIN), a digital library created by the Law Library of Congress for making laws and legal decisions accessible to citizens, legislators, government and private sector officials[3]. Their model grants privileges based on user credentials (sets of typed attributes relating to the same topic or structure, e.g. an *employee* credential may contain an *age*, *address* and *salary* attribute) and object concepts (conceptual hierarchies extracted from the content of an object using a document management mechanism built into GLIN [Holowczak 1997]). Both credentials and concepts are hierarchical. Credentials types (declarations of what attributes are contained in a credential, their type and possible values) are organized into a hierarchy such that a credential type inherits all attributes of the credential type proceeding it in the hierarchy. For example, if an *employee* credential type specified that it contains the attributes *age*, *address* and *salary* and a *international_employee* credential type specified that it contains the attributes *nationality* and *visa*, it would also gain the attributes *age*, *address* and *salary* if *international_employee* was a child of *employee* in the credential type hierarchy.

A simple credential constraint specification language is introduced that allows for the evaluation of user's attribute values (or their assignment to a specific credential) using rudimentary operations ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, $\subset$, $\subseteq$, $\supset$, $\supseteq$, $\not\subset$, $\not\subseteq$, $\not\supset$, $\not\supseteq$, $\in$, $\notin$). Constraint expressions take the form of $X.a$ OP v where X is a variable representing any user in the system, a is an attribute name, OP is an operation and v is a value (for example $X.age > 18$ would specify all users with an age over 18). Constraint expressions can also simply be a credential type to specify all users assigned to a given credential (including children of that credential in the credential type hierarchy). For example, the expression $employee(X)$ would specify all users who are employees. These constraint expressions are used in Access Authorizations to create the access policies of the system. Access Authorizations are tuples consisting of a credential specification (one or more credential expressions joined with *AND* or *OR* symbols), entity specification (denotes the concepts, objects or parts of objects the authorization refers), privilege (a valid operation on an object) and sign (whether the authorization is positive, grants the privilege, or negative, forbids it). Example Access Authorizations are shown in Example 2.9.

Example 2.9. Some Possible Access Authorizations:

- A_1 `= (employee(X), 2016 Income Report, view-all, +)`

  Allows all employees to view the "2016 Income Report".

- A_2 `= (international_employee(X) ∧ X.nationality = Canadian, 2016 Income Report.Canada part, update, -)`

  Forbids international employees from Canada from updating the Canada part of the "2016 Income Report".

- A_3 `= (X.age ≥ 18, Book of Guns ∧ Book of Drugs, view-all, +)`

  Allows any user with an age of 18 or over to view the "Book of Guns" and the "Book of Drugs".

[3]http://www.glinf.org

Adam, et al. also provide details about a supporting system architecture and protocol, discuss an implementation of their model and explain how administrative operations are performed. Later work by the same authors [Ferrari et al. 2002] introduces an authorization system for digital libraries that utilizes this access control and authorization model.

2.3.2 Hybrid Models

Hybrid models of ABAC aim to combine attributes into existing models of access control or to extend the traditional models with identityless or policy-based access control concepts. This includes both early attempts at adding parameterized roles and permissions to RBAC as well as more modern efforts to unify ABAC with alternative access control models such as Relationship-Based Access Control (ReBAC) and Behaviour-Based Access Control (BBAC). Kuhn et al. [Kuhn et al. 2010] describe a number of hypothetical strategies for adding attributes to RBAC:

Dynamic Roles Roles are assigned dynamically based on the user's and environment's attributes, providing identityless access control for RBAC-based systems. Most dynamic role-based hybrid models lack object attributes or a means to dynamically assign permissions to roles, and as such lack the flexibility of ABAC to limit access based on the content of objects (e.g. only allow users to view medical records in which they are the patient). This leads to what has been described as an "explosion" [Jin et al. 2012b] of role-permission assignments or the creation of a large number of private roles.

Attribute-Centric Roles are considered to be just another attribute of a user. No role-permission relation is created and permissions are assigned through policies. If no special consideration for roles is provided in an Attribute-Centric model this could be seen simply as "pure" ABAC modelling RBAC. As this can be seen as equivalent to "pure" ABAC in most cases, it is deprived of the advantages of RBAC (simple administration, auditability, straightforward separation of duties, etc.).

Role-Centric The maximum permission set available in a given session is constrained by attribute-based rules. Constraint rules are used only to reduce permissions available to the user and never expand them (differentiating it from role parameterization). Few details are given about how this strategy may be implemented or if it is different enough from existing models of parameterized RBAC to warrant its own strategy. To date only one published work is known to specifically utilize this strategy [Jin et al. 2012b].

In addition to the strategies described by Kuhn et al., role parameterization [Ge and Osborn 2004; Giuri and Iglio 1997; Abdallah and Khayat 2005] can be seen as a viable option for ABAC-RBAC hybridization. In Parameterized RBAC, permissions (and in some cases roles) are parameterized with conditions that must be met before access is granted to a subject. Often these conditions involve attributes of the object being accessed but may also include attributes of the user and environment (e.g. time).

We categorize the ABAC Hybrids reviewed in this section into the following subcategories:

Parameterized Role-Based Access Control RBAC models based around extending RBAC by parametrizing permissions and/or roles as described in Section 2.3.2.1.

Attribute-Based Role Assignment Models that extend RBAC to add attributes as described in Kuhn et al.'s Dynamic Roles strategy (i.e. assigning roles via user attributes). These models are reviewed in Section 2.3.2.2.

Attribute-Centric Models that extend RBAC to add attributes as described in Kuhn et al.'s Attribute-Centric strategy that would not be classified as "pure" models of ABAC. These models are reviewed in Section 2.3.2.3.

Role-Centric Models that extend RBAC to add permission filtering based on attributes as described in Kuhn et al.'s Role-Centric strategy. To date only Jin et al.'s RABAC [Jin et al. 2012b] is known to exist in this category. This model is reviewed in Section 2.3.2.4.

Unified Models of Access Control Access control models that combine ABAC with with alternative access control models (i.e. non-traditional models) as described in Section 2.3.2.5.

Tables 2.6 to 2.8 summarize and compare the most notable of these hybrid models using similar criteria to the comparison between "pure" ABAC models found in Tables 2.3 and 2.4 (criteria defined in Table 2.2 on page 24).

2.3.2.1 Parameterized Role-Based Access Control

Parameterized Role-Based Access Control (sometimes abbreviated PRBAC [Abdallah and Khayat 2005]) can be seen as an early first step towards ABAC. In PRBAC, permissions normally modelled as object, access mode pairs in RBAC are parameterized with a condition that must be met before the permission is granted to a subject. In Giuri and Iglio's Role Template model [Giuri and Iglio 1997] RBAC permissions are extended with a logical expression referred to as the privilege restriction.

This restriction is evaluated against both the object on which access is requested and the returned value of predefined functions. One example (given in the paper) would be if permissions included "`(delete, PatientRecord, PatientRecord.State = 'discharged')`" then the delete operation would be permitted on any patient record that is in a "`discharged`" state, similarly the permission "`(delete, PatientRecord, today() in [Mon..Fri])`" would permit the delete operation only on week days (Monday to Friday). Additionally, role templates are defined that extend the concept of roles to *"encapsulate and compose parameterized privileges"*. These templates act as a function that takes a set of values (related to the object the role grants access to) and returns a set of parameterized permissions that make up a role. For example, the role template (also taken from the paper) given in Example 2.10 would produce the template instance given in Example 2.11 if the values `prj = "PRJ1"` and `sal = 1000` are used.

Example 2.10. Example role template.

```
R<prj, sal>= role(
 (select, Employee, Employee.project = prj),
 (update, Employee, Employee.project = prj ˆ Employee.salary <sal))
```

Table 2.6: Comparison of Hybrid ABAC Models
An explanation of each column item can be found in Table 2.2.

	Parameterized Role-Based Access Control					Attribute-Based Role Assignment	
	[Ge and Osborn 2004]	[Giuri and Iglio 1997]	[Abdallah and Khayat 2005]	[Lupu and Sloman 1997]	[Fischer et al. 2009]	[Al-Kahtani and Sandhu 2002]	[Shafiq et al. 2005]
Extends	Role Graph Model [Nyanchama and Osborn 1999]	RBAC	FRBAC [Khayat and Abdallah 2003]	RBAC & RBM	RBAC	RBAC	GTRBAC [Joshi et al. 2005]
Identityless	✗	✗	✗	✗	✗	✓	Both
Object Attributes	✓	✗	✓	✓	✗	✗	✗
User Attributes	✓	✓	✓	✗	✓	✓	✓
Environment Attributes	✗	Day of week attribute shown in example but not detailed	✗	Time attribute shown in example but not detailed	✗	✗	Temporal attributes from extended model
Connection Attributes	✗	✗	✗	✗	✗	✗	✗
Mutable Attributes	✗	✗	✗	✗	✗	✗	Mutable trust values
Policy Language	XPath	No policy language formally defined (shown in examples)	N/A	No policy language formally defined (shown in examples)	✓	✓	SAML & X-GTRBAC [Bhatti et al. 2005]
Hierarchical	Hierarchical roles	✗	✗	Hierarchical roles	✗	Hierarchical roles	Hierarchical roles
Trust	✗	✗	✗	✗	✗	✗	✓
Separation of Duties	From extended model	✗	✗	✗	✗	Constraints on use of roles mentioned but not detailed	From extended model
Delegation	✗	✗	✗	✗	✗	✗	✗
Functional Specification	✗	✗	✗	✗	✗	✗	✗
Formal Model	✓	✓	✓	Informal	✓	✓	✓
Administration Model	Does not expand on extended model	✗	✗	✗	✗	✗	✗
Full Model	✓	Definition and evaluation of policies and attributes is only vaguely defined	✓	Lacks details, mostly framework for adding RBM concepts to RBAC	✓	✓	✓

Continued on next page in Table 2.7.

Table 2.7: Comparison of Hybrid ABAC Models (continued from Table 2.6).
An explanation of each column item can be found in Table 2.2.

	Attribute-Based Role Assignment (continued)					
	[Jin and Fang-chun 2006]	[Cirio et al. 2007]	[Cruz et al. 2009, 2008]	[Zhu et al. 2008]	[Wei et al. 2010]	[He et al. 2011]
Extends	RBAC	RBAC	RBAC	RBAC	RBAC	RBAC
Identityless	✓	✓	Both	✓	✓	Both
Object Attributes	✗	✗	✓	✓	✓	✗
User Attributes	✓	✓	✓	✓	✓	✓
Environment Attributes	Temporal attributes	✗	✗	✗	✗	✗
Connection Attributes	✗	✗	✗	✗	✗	✗
Mutable Attributes	✗	✗	✗	✗	✗	✗
Policy Language	ALC(D) [Baader and Hanschke 1991]	Unclear. OWL and SPARQL [Prud'Hommeaux et al. 2008] used for modelling RBAC.	OWL [McGuinness et al. 2004]	Policy language not formally defined	No policy language shown or defined	SWRL [Horrocks et al. 2004]
Hierarchical	Hierarchical roles	✗	Hierarchical roles	Hierarchical roles	Hierarchical roles	Hierarchical roles
Trust	✗	✗	✗	✗	✗	✗
Separation of Duties	✗	✓	✗	✗	✓	✓
Delegation	✗	✗	✗	✗	✗	✗
Functional Specification	✗	✗	✗	✗	✗	✗
Formal Model	✓	Only RBAC modelling formalized	Largely informal	✓	✓	✓
Administration Model	✗	✗	✗	✗	✗	✗
Full Model	✓	Mostly covers modelling RBAC in a an OWL-DL ontology. Few details given on attributes.	✓	✓	Limited details on how constraints and policies are handled or defined	✓

Continued on next page in Table 2.8.

Table 2.8: Comparison of Hybrid ABAC Models (continued from Table 2.7).
An explanation of each column item can be found in Table 2.2.

	Attribute-Centric	**Role-Centric**	**Unified Models of Access Control**		
	[Huang et al. 2012]	[Jin et al. 2012b]	[Han et al. 2009]	[Che et al. 2010]	[Cheng et al. 2014]
Extends	RBAC & ABAC	NIST RBAC [Ferraiolo et al. 2001] & ABAC$_\alpha$ [Jin et al. 2012a]	RBAC, TBAC, & ABAC	ABAC & BBAC	ABAC & UURAC [Cheng et al. 2012]
Identityless	✓	✗	Both	✓	✗
Object Attributes	✓	✓	✓	✗	✓
User Attributes	✓	✓	✓	✓	✓
Environment Attributes	✓	✗	✗	✓	✗
Connection Attributes	✗	✗	✗	✗	✗
Mutable Attributes	✗	✗	✗	Limited. Based on user behaviours.	✗
Policy Language	Informal custom policy language	CPL [Jin et al. 2012a]	XACML	Example policies shown but no language defined.	✓
Hierarchical	✗	Hierarchical roles from NIST RBAC	Hierarchical roles	✗	✗
Trust	✗	✗	✗	✗	✗
Separation of Duties	✗	From NIST RBAC	✓	✗	✗
Delegation	✗	✗	✗	✗	✗
Functional Specification	✗	✓	✗	✗	✗
Formal Model	✓(other than policy)	✓	✓	Largely informal	✓
Administration Model	✗	From NIST RBAC	✗	✗	✗
Full Model	✓	✓	✓	✓	✓

Example 2.11. Resulting role instance for values prk = "PRJ1" and sal = 1000.

```
R<"PRJ1", 1000>= role(
  (select, Employee, Employee.project = "PRJ1"),
  (update, Employee, Employee.project = "PRJ1" ˆ Employee.salary <1000))
```

While the role templates and parameterized permissions described by Giuri and Iglio may provide some advantages over classical RBAC, they do not consider the attributes of the subject, limiting their privilege restrictions to only attributes of objects. This makes policies such as "each student can access their own transcript" difficult to implement without assigning a unique template instance to each student.

The work by Ge and Osborn [Ge and Osborn 2004] towards parameterized roles to support XML databases provides a PRBAC solution that includes both the attributes of subjects and the contents of objects. Ge and Osborn extend the role graph model [Nyanchama and Osborn 1999] to parameterize privileges with XPath-like [Clark et al. 1999] logical expressions that contain variables determined at run time based on attributes defined in a user's session. In an example given in the paper, the parameterized privilege pair "`(//Student[@StudID = param1]/GeneralInfo, update)`" would grant access to update a student's record general info section if the user's student ID attribute matched the student ID in the record. Roles are adapted to support parameterized privileges and the implications of parameterization on inheritance in the role graph is considered. While this extension supports object attributes (limited to the contents of the object) it is only applicable to the narrow domain of restricting access to XML-based databases as opposed to a generic access control solution.

A number of similar PRBAC works have attempted to add logical expression based policies to RBAC permissions including Abdallah and Khayat's PFRBAC [Abdallah and Khayat 2005] (an extension of FRBAC [Khayat and Abdallah 2003]) and Lupu and Sloman's model [Lupu and Sloman 1997] for reconciling Role-Based Management (RBM) and RBAC. Although these and other PRBAC works add aspects of policy- and attribute-based access control to RBAC, they fail to provide the identityless nature of modern ABAC systems. Users (or subjects) still require assignment to roles (in most cases done manually), requiring pre-existing knowledge of both the user and their place in the organization. While sufficient for conventional access control scenarios, identity-based access control like PRBAC fails to provide the flexibility required for emerging computing paradigms including service-oriented architectures (e.g. web services) or dynamic environments as commonly found in cloud computing.

2.3.2.2 Attribute-Based Role Assignment

Models based on Attribute-Based Role Assignment, or "Dynamic Roles" as defined by Kuhn et al. [Kuhn et al. 2010], allocate roles to subjects based on the attributes of the subject and environment at run time. In most cases, administrator created policies are defined via policy languages that relate attributes to constant values (e.g. checking if a user's age is greater than 18) and role assignment is performed when a subject first creates a session with the system based on the outcome of these policies, the user's attributes (e.g. age) and the current state of the environment (e.g. current day of the week). These roles may be limited to a set of possible roles assigned to the user (identity-based) or made totally dependent on attributes with no pre-existing knowledge of the user (identityless).

Al-Kahtani and Sandhu introduce identityless access control concepts into RBAC in their Rule-Based Role-Based Access Control (RB-RBAC) model [Al-Kahtani and Sandhu 2002] by automating the assignment of roles at run time based on a user's attributes. In RB-RBAC, rules defined in a custom policy language determine the set of roles a user is assigned based on attributes provided with the user's credentials. Policy rules take the form of *Attribute_Expression* → *Roles* statements where *Attribute_Expression* is a Boolean statement involving attribute names/values and *Roles* is one or more roles granted if the user's attributes satisfy the attribute expression[4]. Example 2.12 demonstrates three rules that are possible in their policy language. Rule 1 (R_1) grants the *Guest* role to any user between the hours of 9 AM and 5 PM. In rule 2 (R_2), users from Japan or New Zealand who are also 20 years or older are granted the *Adult* role. Finally, in rule 3 (R_3), users from Canada, the USA or Mexico who are 18 years or older are granted both the *Adult* role as well as the *North American* role.

Example 2.12. Example RB-RBAC Policies

```
R1:  (Time IN (900 ..  1700)) → Guest
R2:  (Age ≥ 20) AND (Country IN {Japan, New Zealand}) → Adult
R3:  (Age ≥ 18) AND (Country IN {Canada, USA, Mexico})
       → Adult AND North American
```

Seniority levels are used to denote an attribute's value dominating another value in cases where the order of values is not clear (e.g. strings or sets rather than numerical values), allowing operations such as less than (<) or greater than (>) to be performed on values of any type. The versatility of the model is demonstrated through a number of real life cases; however, the lack of object attributes limits the flexibility of possible policies compared to those possible in most "pure" ABAC models.

A number of approaches [Jin and Fang-chun 2006; Cirio et al. 2007; Cruz et al. 2009, 2008; He et al. 2011] have attempted to use Semantic Web Technologies, such as Web Ontology Language (OWL) [McGuinness et al. 2004], Semantic Web Rule Language (SWRL) [Horrocks et al. 2004] and SPARQL Protocol and RDF Query Language (SPARQL) [Prud'Hommeaux et al. 2008], to both model hierarchical RBAC and extend it with attribute-based dynamic role assignment. Cirio et al. [Cirio et al. 2007] propose both a hybrid RBAC-ABAC model and a supporting framework based on OWL Description Logic (OWL-DL) in which attributes are used to classify subjects into access control roles. While all basic RBAC elements are formalized into a OWL-DL ontology and details for expanding the expressiveness of OWL-DL with SPARQL Protocol and RDF Query Language (SPARQL) are given, Cirio et al. do not fully model the attribute-based aspects of their ontology. Details on how attributes are defined, assigned, related to users or how they may be combined with their framework are not provided. Cruz et al. [Cruz et al. 2009, 2008] describe a "Constraint and Attribute Based Security Framework for Dynamic Role Assignment" focused partly on using a user's physical location for role assignment. In this approach, predefined roles can have both a previously known sets of users as well as users dynamically assigned based on the content of their attributes and policy set on role assignment (referred to simply as constraints in the work). Rather than employing a policy

[4]The grammar of the policy language presented in [Al-Kahtani and Sandhu 2002] allows for more flexible policy rules that include more complex constrains, restrictions and role combinations. However, the paper leaves most of these to future work/extensions.

language like most ABAC works, constraints are defined as attribute name, constraint pairs (e.g. <*Age*, $\geq$ *18*>) that are assigned directly to roles to limit their assignment. Semantics for role inheritance and constraint dominance are given in addition to a description of an OWL-DL ontology based prototype. Finally, both He et al. [He et al. 2011] and Jin & Fang-chun [Jin and Fang-chun 2006] have also produced semantic web-based RBAC models that add elements of ABAC. Both works represent and provide a means to reason about hierarchical RBAC in description logic, He et al. using SWRL [Horrocks et al. 2004] and Jin & Fang-chun using ALC(D) [Baader and Hanschke 1991]. Both also use attribute-based policies for role assignment. The main difference between these models is the limitations put on attributes and support for separation of duties; He et al. limit attributes to user credentials that have been verified by a trusted third party (a process described in their accompanying architecture) and support classical RBAC separation of duties, while Jin & Fang-chun allow temporal attributes in addition to the attributes of subjects but lack any notation of separation of duty style constraints.

Shafiq et al. propose an agent-based framework [Shafiq et al. 2005] for attribute-enhanced RBAC in distributed environments that extends the Generalized Temporal Role-Based Access Control (GTRBAC) model [Joshi et al. 2005]. In this framework, users are both directly assigned roles before hand and allowed to request additional roles at run time based on their self declared attributes and the amount of trust a service provider has in those attributes (determined partly based on additional credentials submitted by the user). In addition to allowing temporal constraints on activating roles (e.g. only allowing the role employee to be activated between 9AM and 5PM), the framework also allows constraints to be placed on the duration a role can be enabled in a given time interval, defined either for a single session or a total duration of all sessions in which the role is active. The X-GTRBAC [Bhatti et al. 2005] XML-based policy language is extended to support SAML-based assertions and attribute-based authorizations used in the framework. While this work presents a novel extension to GTRBAC to support hybrid ABAC in cases where a single trusted attribute authority may not be available, like RB-RBAC it is also omits support for object attributes, limiting the expressiveness of possible policies.

A number of comparable models aim to provide analogous support for attribute-based role assignment for web services and service-oriented environments, including the work done by Zhu et al. [Zhu et al. 2008] and Wei et al. [Wei et al. 2010]. Zhu et al. put forward their General Attribute-Based Role-Based Access Control (GARBAC) model aimed at web services while Wei et al. introduce their Attribute and Role-Based Access Control (ARBAC) model aimed at service-oriented environments. Both models provide hybrid ABAC for service-oriented architectures and support a similar set of features including object attributes and hierarchical roles. While these models add object attributes (something lacking in other models in this subcategory) they lack formal definitions of the policy language being used or the semantics behind it. It is also left unclear how object attributes might be used in role assignment policies in practice if assignment takes place before requests on specific objects are performed (in GARBAC a constraint on role-permission assignment is hinted at but only shown partly in an example case study).

2.3.2.3 Attribute-Centric

Models based on the "Attribute-Centric" strategy, as defined by Kuhn et al. [Kuhn et al. 2010], have the characteristic of incorporating attributes into RBAC model roles as just another attribute of the user and not necessary a separate access control entity onto which permissions are assigned. Instead, permissions are assigned based on evaluating policies relating attributes of users, the environment, objects, etc., with each other and constant values. If no special consideration for roles is given, this is equivalent to the "pure" ABAC models described in Section 2.3.1 which can be seen as a more generalized access control model than RBAC (as it is possible to emulate RBAC configurations in ABAC policies). If special consideration for roles is provided, such as using role-based separation of duties, the model is considered to be an ABAC-RBAC hybrid and described in this section.

The most notable Attribute-Centric work that does not fall into the category of "pure" ABAC is Huang et al.'s "a framework integrating attribute-based policies into role-based access control" [Huang et al. 2012] that models RBAC on two levels. A front end (or "aboveground") level presents itself as a traditional RBAC model extended only with environmental attributes (applied to both user-role and role-permission assignments) and a back end (or "underground") level emulates the simplistic RBAC front end using attribute-based policies. This departmentalizing allows routine access control operations and auditing/review to be performed on the simpler RBAC front end, while still allowing the more complex administration and fine grained attribute-based policies to be created in the ABAC back end.

Underground level policies are divided into two categories: Role-permission assignment policies, that determine assignment of permissions to roles and user-role assignment policies that determine the assignment of users to roles. Both types of policies are specified using First Order Logic (FOL) expressions that follow structures shown below:

Role-Permission Assignment Policy Structure

```
rule_id {
 target {
  role_pattern;
  permission_pattern {
   operator_pattern;
   object_pattern;
  }
  environment_pattern;
 }
 condition;
 decision.
}
```

User-Role Assignment Policy Structure

```
rule_id {
 target {
  user_pattern;
  role_pattern;
  environment_pattern;
 }
 condition;
 decision.
}
```

Where `patterns` are FOL expressions that define a set of environmental states, set of roles, set of users, set of object, etc. as appropriate and comprise the `target` of the rule (the access control entities to which this rule applies). The `condition` is a FOL expression that defines conditions that must be met for the role or permission to be assigned and the `decision` defines the exact role or permissions assignment that will be made. Example 2.13 shows a user-role assignment policy that grants any role of type "employee" to any user (as no user pattern is given) located in London. The granted roles are only valid in environments matching the environmental pattern specified. In this case, only on weekdays and while the system mode is set to “normal”.

Example 2.13. Example Policy in Huang et al.’s Framework

```
rule: {
 target: {
  role_pattern(r): r.type = "employee";
  environment_pattern(e): {
    Time = "Weekday"
    and Mode = "Normal" }
 }
 condition: {
  u.location = "London";
 }
 decision: add (u,r,e) in URAe.
}
```

While this dual level model simplifies administrating a large scale ABAC system, this benefit is only maintained if policies of the back end ABAC model conform to those reviewable in a standard RBAC framework. Back end policies that grant roles based on non-identity related attributes (e.g. location, time, etc.) rather than limit activation of or put constraints on previously assigned roles can easily lead to issues when attempting to determine the set of users who have access to a given role or permission (as is the case with most ABAC systems). This forces the role/policy engineer to choose between creating an identityless access control system or one which is easily auditable.

2.3.2.4 Role-Centric

Jin et al.’s Role-centric Attribute-Based Access Control (RABAC) [Jin et al. 2012b] extends the NIST RBAC model [Ferraiolo et al. 2001] to create the first attempt at a formal Role-Centric RBAC-ABAC hybrid model. RABAC follows Kuhn et al.’s approach [Kuhn et al. 2010] of reducing the number of permissions available to a subject in a traditional RBAC session based upon the current value of attributes (in this case only user and object attributes). Permission filtering policies, defined in a custom Common Policy Language (CPL) [Jin et al. 2012a] based language, are used to reduce the maximum permission set in a given session by checking each permission against all applicable filtering policies. The applicability of each policy is determined by a secondary “condition” policy assigned to each filtering policy that determines if it should be applied to a given permission based on the attributes of the object. This method is used to constrain permissions without significantly modifying the NIST RBAC model (only the set of permissions available to a subject in a given session are effected) enabling other concepts such as separation of duties or the role hierarchy from the NIST model to be directly applied to RABAC without modification.

While this work does provide a first attempt at a role-centric model, it is unclear if it poses a significant benefit over preexisting models of PRBAC. Both offer an identity-based solution that constrains role-permission assignment, the main difference being that PRBAC changes the process of the role-permission relation such that permission assignment is determined at run-time while RABAC keeps the relation unchanged and filters permissions out during session creation. Jin et al. argue that this difference enables RABAC to make use of the NIST RBAC administrative model while PRBAC models would require new and more complex administration models.

2.3.2.5 Unified Models of Access Control

We define *Unified Models of Access Control* as any models of access control that attempt to combine two or more non-traditional models of access control into a single unified model. For the purposes of this section, only models that include ABAC are considered. Cheng et al. attempt to combine Relationship-Based Access Control (ReBAC) with ABAC in their UURAC$_A$ model [Cheng et al. 2014] by extending the User-to-User Relationship-Based Access Control (UURAC) [Cheng et al. 2012] model. ReBAC-based models provide access control for Social Network Systems (SNS) based on a subject's relations with other users and entities in the social network. For example, a user may create an access policy to limit access to viewing their profile to only friends or friends of friends (i.e. limiting access to the profile to users with a user-to-user relationship depth of 1 or 2 from the profile owner on the social graph). Cheng et al.'s UURAC$_A$ adds attributes to both the nodes (users and resources) and edges (relationships) of the social graph, representing attributes of users, resources and relations (type, weight, trust, etc.). A custom policy language (based on the language from UURAC) enables users to restrict access to owned resources based on a combination of attributes and relations. The following example (Example 2.14) policies (taken from [Cheng et al. 2014]) restrict access to a profile based on users who share at least five common friends who are students (P$_1$), restrict access to a profile to friends in common with "Bob" (P$_2$) and restrict access to a photo to users who are within 3 hops of the owner on the social graph with a minimum trust value of 0.5 at each hop (P$_3$).

Example 2.14. Example UURAC$_A$ Policies

$$P_1: \langle \textit{profile\_access}, (u_a, ((ff, 2) : \exists[+1, -1], \textit{occupation}(u) = \text{“}\textit{student}\text{”}, \textit{count} \geq 5)) \rangle$$

$$P_2: \langle \textit{profile\_access}, (u_a, ((ff, 2) : \exists[+1, -1], \textit{name}(u) = \text{“}\textit{Bob}\text{”}, \_)) \rangle$$

$$P_3: \langle \textit{read}, \textit{Photo1}, (u_a, ((f*, 3) : \forall[+1, -1], \textit{trust}(r) \geq 0.5, \_)) \rangle$$

While UURAC$_A$ successfully adds attributes to UURAC, there are some possible privacy concerns resulting from allowing end users to define their own attribute-based policies (something that is not unique to UURAC$_A$ but any ABAC model that allows users to create policies to protect their own resources/objects). For example, if a user, Alice, has a private profile on a SNS and an attacker, Eve, wishes to obtain some private information from that profile that is also an attribute describing Alice (e.g. location, age, gender, occupation, etc.). Eve could generate a large number of resources that would be appealing to Alice to view (e.g. a link to a picture with the text "Is this you in this picture?") and protect each resource with a policy that contains a guess at the value of one of Alice's attributes (e.g. $(\textit{name}(u) = \text{“}\textit{Alice}\text{”}) \wedge (\textit{age}(u) = 18)$, $(\textit{name}(u) = \text{“}\textit{Alice}\text{”}) \wedge (\textit{age}(u) = 19)$, $(\textit{name}(u) = \text{“}\textit{Alice}\text{”}) \wedge (\textit{age}(u) = 20)$, etc.). Alice would only be able to access the resource with the correct value and Eve would be able to determine this value by checking which resource is accessed. For example, if the set of resources were posts containing a link, each to a different image on Eve's website. Eve could determine the value of the attribute by matching the accessed image to the policy used to protect the accessed resource. This sort of attack could also be conducted more efficiently by using ranges of values for the attributes Eve is guessing at (e.g. $(name(u) = \text{“}Alice\text{”}) \wedge ((age(u) > 10) \vee (age(u) < 20)))$ to narrow down the value with fewer resources generated.

Che et al.'s Behaviours and Attributes-Based Access Control (BABAC) [Che et al. 2010] attempts to unify Behaviour-Based Access Control (BBAC) and ABAC to provide a novel access control solution for network virtualization. In BABAC, user behaviours (a single or sequence of actions performed by a user) are quantized and divided into three categories; Time-Lasting Behaviour (a single persistent action that last for a fixed amount of time), Instant Behaviour (a single action that happens instantly and has no associated length of time), and Multi-Action Behaviour (A combination or sequence of Time-Lasting and Instant behaviours). These behaviours are then used in combination with user and environment attributes to define access control policies that restrain access to resources both before and after permissions are assigned (e.g. a user's access to a resource could be revoked if they spend too much time performing a single action). The BABAC revocation policy in Example 2.15 (from [Che et al. 2010]) revokes read access to the resource “FinancialPlan” if the user views the resource for more than 60 minutes, attempts to perform an illegal copy operation or more than 3 users are trying to access this resources at one time. The time-lasting behaviours (TB), instant behaviours (IB), and multi-action behaviours (MB) that will be used in the policy are specified before the revoke policy expression.

Example 2.15. Example BABAC Revocation Policy

```
Resource = "FinancialPlan"
Action = "Read"
TB = "TotalViewTime"
IB = "PerformIllegalCopy"
MB = "TotalSeveringUser"
Revoke(U,R,A) ⇐ { TotalViewTime(U) ≥ 60 minutes
  ∨ PerformIllegalCopy(U) = true
  ∨ TotalSeveringUser ≥ 3 users }
```

To support access requests between independent virtual networks, user attributes are divided into three types; Global Attributes (user attributes obtained from a virtual network independent global attribute authority trusted by all virtual networks), Intra-domain Attributes (user attributes defined locally by an individual virtual network that access is currently being requested upon), and Trust-domain Attributes (user attributes imported from remote virtual networks that are trusted by the current network upon which access is currently being requested). Example 2.16 shows how these attributes may be used in a BABAC policy to grant access to a resource (the same financial plan as in Example 2.15). In this case, a user is allowed read access if they have a global security level of 5 or greater, have a job title of “junior-manager” in the local network or have a job title of “senior-manager” in a trusted network and are not located in department C of a trusted network.

Example 2.16. Example BABAC Granting Access to a Resource

```
Resource = "FinancialPlan"
Action = "Read"
GAttr = "SecureLevel"
```

```
IAttr = "JobTitle", "Location"
TAttr = "JobTitle", "Location"
Allow(U,R,A) ⇐ { SecureLevel(U) ≥ 5
∧ (JobTitle(U) ≥ IAttr(junior-manager)
   ∨ JobTitle(U) ≥ TAttr(senior-manager))
∧ Location(U) ≠ TAttr(dept.C) }
```

One last notable effort, is Han et al.'s [Han et al. 2009] work towards a united access control model that combines ABAC, RBAC and Task-Based Authentication Control (TBAC). In Han et al.'s united model, TBAC is extended with attribute-based constraints (limited to user and object attributes) in addition to hierarchical role-based assignment of task permissions. Permissions are divided into Executing (permission to execute a task), Supervising (permission to initiate, approve, dispense, or administrate task execution) and Invoking (permission to initiate task request and acquire the result) permissions which are granted by roles. ABAC is used largely for negotiating identityless role assignment with external users and functions similarly to attribute-based role assignment.

While unified models provide interesting new takes on existing non-traditional models, they are often limited in their applicability to real world access control scenarios, instead targeting niche access control scenarios or domains. UURAC$_A$'s application is limited to SNS, BABAC to network virtualization and Han et al.'s united model to systems in collaborative commerce. Additionally, combining models often leads to increased complexity such as is the case in Han et al.'s united model where administrators are required to deal with attributes, policies, role assignments, role hierarchies, workflows and tasks for both internal and external users; all in a single access control system. While this provides a large number of fine grained configuration points, it is questionable how manageable or auditable real world implementations would be, especially in systems with a large number of access control entities.

2.4 Open Problems

As ABAC research is still largely in its infancy, the list of open problems related to ABAC systems and implementations is extensive. The majority of these problems stem from the increased complexity attribute and policy-based access control introduces for the sake of increasing the flexibility and generality of access control policies. While hybrid ABAC models and frameworks aim to remedy these issues by extending proven traditional models, this is often done at the cost of flexibility or removing the identityless nature of ABAC. This section outlines the most common problems identified and discussed in the recent literature (namely the works reviewed in Section 2.3) relevant to ABAC and to a lesser extent, policy-based access control in general.

2.4.1 Foundational Models

One frequently discussed issue [Jin et al. 2012a; Hu et al. 2013; Servos and Osborn 2014] is the lack of an agreed upon reference and/or foundational model of ABAC. While a large number of ABAC models have been published, they have predominantly been domain specific and limited to a particular use case (e.g. web services) or hybrid models that lack the versatility of "pure" models. Of the generalized models discussed in Section 2.3.1, only three [Jin et al. 2012a; Servos and Osborn 2014; Zhang et al. 2005] are both formal and full models, none of which have garnered mainstream acceptance as "the standard" model of ABAC.

To date, the most frequent works cited as "the model of ABAC" have been XACML, Wang et al.'s logic-based framework for ABAC [Wang et al. 2004] and Yuan & Tong's ABAC for web services [Yuan and Tong 2005]. However, these works are problematic as foundational models for a number of reasons. As XACML is simply an access control policy language, it lacks any kind of formal model of ABAC despite its support for attributes, making it at best only one component of a larger model. Wang et al.'s logic-based framework, provides a start towards a generic foundational model but mostly concentrates on modelling policies and their evaluation and can not be seen as a full model of ABAC. Yuan & Tong's ABAC model for web services, while an early effort and the basis for several other models [Kerschbaum 2010; Xia and Liu 2009], is simplistic and specific to a limited domain. Perhaps the most promising, but yet to be completed or published, work is the purported effort at NIST towards a formalized family of ABAC models. During the NIST Attribute Based Access Control Workshop held on July 17, 2013, limited details on the "Framework of ABAC models" were presented by David Ferraiolo that defined four families of ABAC models; $ABAC_{rule}$, $ABAC_{rule-hier}$, $ABAC_{rel}$ and $ABAC_{rel-history}$. Unfortunately, to date, few details and no formal definitions are available for these models (the only source being an unrefereed set of presentation slides [Ferraiolo 2013]).

Beyond model adoption or creation by a standards organization, a possible solution may lie in the suggestion of Barker [Barker 2009] for access control research to avoid "developing the next 700 particular instances of access control models" and instead focus on unifying meta-models. A meta-model of ABAC, or perhaps all policy-based access control in general, could provide a unified model for describing and reasoning about ABAC without necessitating the need for creation of new models for each small extension of the concept.

2.4.2 Emulating and Representing Traditional Models

It has been claimed that ABAC is a more general model of access control as it is capable of emulating the traditional models [Chadwick et al. 2003; Jin et al. 2012a; Lang et al. 2009; Servos and Osborn 2014; Park and Sandhu 2004]; however, as of now this has only been demonstrated in the literature in a largely informal and shallow manner. The work by Jin et al. [Jin et al. 2012a] has presented the most formal effort to date, demonstrating how ABAC$_\alpha$ can be constrained to model DAC, MAC and hierarchical RBAC. However, only a single possible representation is given for each classical model (a number of which assume a partially ordered set may be used as an attribute's value) and the separation of duty constraints of RBAC are not modelled. A deeper exportation and evaluation of the different possible methods of representation are required to both develop best practices for aiding in the transition to ABAC (e.g. converting existing traditional systems to ABAC systems) and formally proving that ABAC can model all possible DAC, MAC and RBAC-based policies.

2.4.3 Hierarchical ABAC

In hierarchical RBAC, the role hierarchy allows for roles to be related in a way that more closely resembles that of actual organizations. This allows for more simplistic administration, both in terms of role engineering and reviewability of existing role-based policies. Most "pure" models of ABAC; however, lack this type of inheritance and expressiveness. While a role can be easily modelled as a single attribute of a subject, this simplistic representation is unable to emulate the hierarchical nature of RBAC without allowing for complex data types in an attribute's value (as is done in Jin et al. [Jin et al. 2012a] ABAC$_\alpha$) or unmaintainably complex policies. A more simplistic means of providing hierarchical administration is required for "pure" ABAC to be competitive with RBAC and hybrid models.

A possible solution may be found in "attribute user groups" [Servos and Osborn 2014] (as described and introduced in Chapter 3), hierarchical groups that inherit sets of attributes from their parent groups and allocate these attributes to their members (similar to how roles in hierarchical RBAC could be seen as allocating permissions to the role's membership). This technique could also be applied to objects and other access control entities onto which attributes may be assigned. Another approach is to allow attributes to have inheritance relations directly with other attributes, such that a child attribute supersedes the parent attribute in policies. For example, if both the attributes "cs_faculty" and "cs_graduate_student" are children of the attribute "cs_department", being assigned "cs_faculty" or "cs_graduate_student" would fulfil a policy requiring a user to be assigned the "cs_department" attribute. This is similar to the attribute hierarchies described in Wang et al.'s ABAC framework [Wang et al. 2004] as well as other models, but potentially limits the usefulness of ABAC as attributes no longer have values (instead each attribute hierarchy could be seen as a single attribute with members being the possible values for the attribute).

2.4.4 Auditability

An important aspect of access control for both legal and security reasons is the ability to easily determine the set of users who have access to a given resource or the set of resources a given user may have access to (sometimes referred to as a "before the fact audit"). In RBAC, this is relatively straightforward, normally just requiring the system to calculate the union of the set of effective privileges from each role the user is assigned. However, in ABAC this is considerably more complicated [Hu et al. 2013]. As ABAC is an identityless access control system and users may not be known before access control request are made, it is often not possible to compute the set of users that may have access to a given resource. Even in cases where the identities of all users and their assigned attributes are known, it can still be difficult to efficiently calculate the resulting set of permissions for a given user as all objects would need to be checked against all relevant policies.

To date, this has largely been addressed with hybrid ABAC models that use attributes simply for role assignment [Al-Kahtani and Sandhu 2002; Shafiq et al. 2005; Jin and Fang-chun 2006; Cirio et al. 2007; Cruz et al. 2009; Zhu et al. 2008; Wei et al. 2010] (allowing administrators to at least know what roles grant what permissions) or to put constraints on the permissions assigned to a role [Ferraiolo et al. 2001; Ge and Osborn 2004; Giuri and Iglio 1997; Abdallah and Khayat 2005; Lupu and Sloman 1997] (favouring an identity-based approach). As these methods use hybrid strategies, they come with the disadvantages of the hybrid models they use (i.e. namely loss of flexibility and identityless access control). ABAM [Zhang et al. 2005] is one of the few "pure" ABAC models that provides some level of auditability by restricting subjects to only possibly being assigned permissions in a predefined access matrix; however, it accomplishes this at the cost of being identityless and requires users to be known and properly labelled in the access matrix.

It is important that more complete and efficient methods of auditing "pure" ABAC systems be developed to enable administrators to demonstrate compliance with specific regulations and directives that require before the fact auditing. Without this ability, ABAC will likely be unusable in cases where legal or industry regulations prohibit systems that rely solely on after the fact auditing techniques.

2.4.5 Separation of Duties

Separation of Duties (SoD) is the notion that multiple persons should be required to complete a sensitive task to limit the potential for both error and fraud. In RBAC, this is supported through static SoD, where subjects are prohibited from being assigned conflicting roles, and dynamic SoD, where subjects are prohibited from activating conflicting roles in the same session [Ferraiolo et al. 2001]. However, in ABAC, application of this concept has been largely unexplored and left to future work. It is still unclear to what or how SoD type constraints might be applied to ABAC models and if additional constraints beyond those possible through policy languages are required.

Alipour & Sabbari [Alipour and Sabbari 2012] attempt to solve this problem by introducing "can't_perform" rules that restrict a subject from performing certain actions (operations) on specified resources. This solution is problematic; however, in that it requires knowledge of both the subject and their possible conflicts of interest beforehand. Bijon et al. propose an attribute-based constraint specification language, Attribute-Based Constraints Specification Language (ABCL) [Bijon et al. 2013], that allows constraints to be placed on both attributes and attribute assignments. They demonstrate how this language may be used to specify SoD style constraints and validate its usefulness through a number of use cases. While this work may be part of a viable solution, it merely defines a language for representing constraints and lacks a formal model or framework for their use. Finally, a common solution is to use the SoD constraints from RBAC in hybrid ABAC models that include roles [Jin et al. 2012b; Shafiq et al. 2005; Cirio et al. 2007; Wei et al. 2010; Han et al. 2009]. However, as with other uses of hybrid ABAC, it comes at the cost of flexibility or the identityless nature of ABAC.

2.4.6 Delegation

Delegation is a frequently desired access control feature that allows one subject to temporarily delegate their access rights to a more junior (in terms of access rights) subject. In RBAC research [Barka and Sandhu 2000a,b] this is often accomplished by enabling delegation of assigned roles under certain predefined constraints and revocation conditions, but has also been expressed in terms of partial permission delegation [Wang and Osborn 2011; Zhang et al. 2003; Wang and Osborn 2006], in which a delegator creates and delegates a temporary role composed of a subset of their delegatable permissions. While delegation has been partially addressed in terms of attribute-based encryption [Waters 2011; Servos et al. 2013], few efforts to date have been made to apply a delegation model to ABAC.

Such a model of delegation could be applied to both delegation of attributes between users and delegation of resulting permissions granted by policies. Delegation of attributes could be partially supported through the use of X.509 attribute certificates [Farrell and Housley 2002; Farrell et al. 2010]; however, this requires potentially lengthy certificate chains to be transmitted as part of a user's attribute-based credentials and could also lead to privacy concerns when sensitive attributes are involved. Moreover, attribute certificates are largely an implementation detail rather than a formal part of a delegation model. Dynamic delegation of permissions is more complex as attribute values (particularly for environment attributes like time) may frequently change resulting in different permission assignments. Allowing delegation of granted permissions may require constant evaluation of relevant policies to ensure permissions are revoked when the delegator's access is removed due to a change in attributes, an approach that is both complicated and inefficient.

2.4.7 Attribute Storage and Sharing

When multiple attribute sources are used in an ABAC system (e.g. using attribute authorities from different organizations in a distributed system) complications can arises in terms of both evaluating the trustworthiness of attributes and ensuring that differing attribute sources are using compatible attributes (e.g. using the same namespace and data type for common attributes). The issue of trustworthiness is often dealt with by relying on pre-existing trust relations negotiated between organizations before access control takes place; however, in peer-to-peer scenarios this can be vastly more complicated. Shafiq et al. [Shafiq et al. 2005] offer a potential solution in their hybrid ABAC model that includes a trust evaluation and negotiation framework that both provides a trust assessment of claimed attributes and a means to dynamically establish trust between collaborating organizations. Lee et al. [Lee et al. 2008] propose an "attribute aggregation architecture" where attributes are gathered from neighbouring peers and evaluated using a reputation-based trust scheme in which "each peer decides its reputation about other peers based on its own experiences, and the trustworthiness of a peer is evaluated with the assist of aggregated reputation". It is possible that Shafiq's, Lee's other research in dynamic trust negotiation could be easily applied to "pure" ABAC models; however, most work in this area has assumed attributes are derived from a trusted source.

Ensuring attributes from different sources are compatible would likely require a commonly accepted namespace or ontology of attribute names or alternatively some means of mapping attributes to equivalent representations (as suggested in [Hu et al. 2013]). For example, if one organization's attribute store uses the name "job_title" and another "role" to describe the same attribute it would be difficult to create policies that are applicable to members of both organizations without a detailed mapping between the two sets of attributes or complex policies that take into account the differences in attribute composition in each store. A secondary issue in attribute sharing is ensuring the confidentiality of sensitive user attributes. This is particularly a concern when ABAC systems are used in domains such as health care where leaking attributes about a user or object could be potentially compromising. Current work related to attribute privacy or confidentiality has largely been limited to attribute-based encryption applications but some efforts have been made towards generic privacy preserving attribute sharing protocols

[Camenisch et al. 2010; Ardagna et al. 2010; Esmaeeli and Shahriari 2010; Zhang et al. 2013].

2.4.8 Scalability

One of the important considerations before adopting ABAC (as described in the NIST Guide to ABAC Definition and Considerations [Hu et al. 2013]) is the scalability of ABAC systems. Unlike traditional access control technologies, such as RBAC, that have a proven track record in being adopted in large scale real world systems, ABAC is still largely unproven in terms of practical scalability. ABAC requires complex interactions between access control components that may be distributed among different network resources or even across organizational boundaries. In large systems with thousands of users, permissions, and policies, it is unclear how manageable ABAC solutions would be both in terms of administration and physical computing resources required. Real world case studies of large scale systems utilizing ABAC concepts are required to determine the feasibility and usability of ABAC in such scenarios.

2.4.9 Administration and User Comprehension

A frequently overlooked aspect of ABAC is the "human aspect" or how usable such systems may be for users, access control administrators and policy engineers. Lee & Winslett [Lee and Winslett 2006] discuss the human factor challenges related to ABAC solutions and identify a number of open problems in ABAC research related to administration and usability. They describe the three main challenges as "Access Control Comprehension", "Technology Management" and "Policy Specification and Maintenance".

Lee & Winslett characterize "Access Control Comprehension" as the end user's ability to comprehend the access control decisions made regarding their access requests. In the classical models, access control decisions are relatively straightforward (e.g. in RBAC, users are either members of a role with the effective permissions they desire or not). However, in ABAC, decisions may be the result of complex policies that not only involve the attributes of the user but attributes of other, frequently changing, access control entities. Without sufficient understanding of both ABAC and the existing policies contained in the system, access decisions may seem arbitrary if not entirely magical from an end user perspective. Lee & Winslett point to efforts by Yao et al. towards visualization of such decisions [Yao et al. 2005] as a first step towards a potential solution.

"Technology Management" concerns a user's ability to manage their access control credentials. In ABAC, subject credentials can be rather complex, consisting of technologies such as cryptographic credentials, X.509 certificates and attribute sources from multiple distributed attribute stores. Lee & Winslett point to the research by Whitten & Tygar [Whitten and Tygar 1999] in which users had extreme difficulty managing PGP certificates for signing and encrypting e-mails to argue that end users of ABAC systems will have similar if not more extreme difficulties. This burden is worsened in systems that rely on end users to select the subset of attributes to be activated in a given session. While the solution to this problem likely lies in automating credential management, this has been largely unexplored in relation to ABAC and warrants further study.

"Policy Specification and Maintenance" addresses challenges related to the increased complexity inherent in ABAC administration and policy engineering. To date, almost no ABAC models provide complete (or even partial) administration models while at the same time requiring administrations and engineers to provide policies composed in complex XML-based policy languages such as XACML. Furthermore, the potentially distributed nature of ABAC means administration is no longer centralized but divided among multiple policy administration points and attribute stores. This significantly raises the training and education requirements for competent administrative users as well as hindering their ability to review current configurations for security issues. Potential solutions may be found in analysis tools that allow users with limited knowledge of mathematical or Boolean logic to create and evaluate realistic access control policies, in automated tools for mining ABAC policies [Xu and Stoller 2014, 2013, 2015] and in new administrative access control structures such as hierarchical attribute user and object groups [Servos and Osborn 2014] (introduced and described in Chapter 3).

2.4.10 Formal Security Analysis

While a number of works have sought to provide tools to analyze the security and safety of the traditional models (namely RBAC) [Li and Tripunitara 2006; Sasturkar et al. 2006; Stoller et al. 2007] and the policies they enforce, similar efforts for ABAC are still in their infancy. The most relevant efforts to date (e.g. [Bryans 2005; Lin et al. 2010; Fisler et al. 2005; Kolovski et al. 2007]) have focused on reasoning about and analysing access control policies that may support attribute-based concepts independently of a formal access control model (e.g. policies written in XACML). Although many of these concepts and tools can be applied to the policies supported by ABAC models (particular if they are in a standardized policy language like XACML), they alone can not provide a full security analysis of a given ABAC model without taking into consideration the properties of the underlying model and the way in which policies are combined and enforced.

A sensible starting point for future ABAC focused security analysis work may be found in adapting the techniques used for RBAC such as those employed by Li & Tripunitara [Li and Tripunitara 2006]. Li & Tripunitara use security analysis techniques [Li and Winsborough 2003] to view RBAC as a state-transition system in which state changes occur via administrative operations, with the goal of determining if undesirable states are possible. Whilst they primarily use this state-transition system to explore security problems resulting from RBAC administration, a number of the queries they define on a given system state could be adapted for analysing ABAC systems. In particular, the following queries could be of interest if attributes are considered in place of roles:

Simple Safety If a state exists where a given (presumably untrusted) user can gain membership in a given role only intended for trusted users. A negative result would imply that the system is safe.

Simple Availability If a given permission is attainable in every possible state to a given (presumably trusted) user. A positive result would imply that the permission is always available to the user.

Bounded Safety If in every possible state, only a given subset of (presumably trusted) users can obtain the given permissions. A positive result would imply that the system is safe.

Liveness Whether a given permission is always accessible to at least one user. A negative result (i.e. that the permission is always accessible) would imply the liveness of the permission holds in the system.

Mutual Exclusion If there exists no possible state where a user can be a member of two distinct roles (r_1 and r_2). A positive result would imply that roles r_1 and r_2 are mutually exclusive.

Containment Whether in every reachable state any user who has a given permission is a member of a given role. A positive result would imply that safety property is held (i.e. that all holders of a given permission are also in a given role) and an availability property is held (i.e. that a given permission is available to all members of a given role).

Adapting such queries to ABAC systems is challenging due to the increased flexibility provided by attribute-based policies and its identityless nature in which users may not be known until runtime. This posses similar problems as faced when auditing ABAC systems (as discussed in Section 2.4.4), namely that efficiently calculating the result of such queries is difficult when a large number of policies and attributes are present in a system. Rather than simply considering system states created by a combination of users, roles and permissions (as in RBAC), analysis of ABAC system would have to account for all possible combinations of attributes (including possible combinations of values for each individual attribute), policies and permissions. Leading to a drastically larger state space.

2.5 Conclusions & Future Work

2.5.1 ABAC Survey

This chapter has introduced a taxonomy of current areas of ABAC and Policy-Based Access Control (PBAC) research, provided a literature review of current attempts at formalizing ABAC models, and identified a number of open problems in the literature. The taxonomy introduced in Section 2.2 subdivides the current body of ABAC related research into related categories that are useful when discussing and comparing recent efforts. The review of "pure" and hybrid ABAC models in Section 2.3 provides one of the most comprehensive summaries of existing academic work towards ABAC model creation and has proven useful in identifying a number of areas for future work. The open problems examined in Section 2.4 serve as potential starting points for new research efforts.

As the literature surveyed in this work covered a number different types of ABAC models in breadth, there is still room for future survey efforts directed at covering specific categories or aspects of models in depth. An in-depth comparison and analysis of how current models represent attribute-based policies, for example, would be of benefit to the community. As would a more in-depth look at a specific subcategories of models (e.g. a longer review of Pure General ABAC Models). Reviews of non-model related attribute topics could also be of interest, such as attribute mining, attribute storage and sharing, attribute confidentiality and supporting model independent architectures.

Since the date this survey was conducted (October 20th, 2014) there have been a number of publications relating to attribute-based models of access control. Some of these are touched upon in Chapter 7 Section 7.3.1, but due to the rapid rate of developments in this field there is a clear need for ongoing reviews and systematizations of the literature in this area. Further surveys targeting ABAC models published after this survey could be useful to cover more recent advancements.

2.5.2 Addressing the Open Problems

The open problems (Section 2.4) identified in this chapter are a potential road block for the adoption and standardization of ABAC. A number of these problems are addressed by the subsequent chapters in this thesis, while others are left to future work or are currently being pursued by others (some of which are discussed in Chapter 7).

Chapter 3 addresses the lack of hierarchical ABAC models (an issue discussed in Section 2.4.3) by introducing HGABAC, a formal ABAC model with support for hierarchical user and object attribute groups. HGABAC also attempts to address the problem of administrator and user comprehension (an issue discussed in Section 2.4.9) via attribute groups. Attribute groups allow for the assignment of multiple related attributes to a user or object via membership in a group, helping administrators to assign or modify related collections of attributes at a time rather than individually. It is also shown that the traditional models (RBAC, DAC, and MAC) can be emulated using these groups (aiding with the issue of representing the traditional models as mentioned in Section 2.4.2).

Chapter 5 provides a supporting architecture for HGABAC that fills in the gaps between model and real world implementation, providing details about attribute storage, sharing, transmission and namespaces (open problems described in Section 2.4.7). Some investigation into the scalability of the system is also provided (addressing potential issues of scale brought up in Section 2.4.8) and new attribute certificate format is defined to enable the "off-line" sharing and proof of attribute ownership.

Chapter 4 and Chapter 6 are directed at the problem of delegation in ABAC (a problem defined in Section 2.4.6). Chapter 4 outlines a number of potential strategies for incorporating delegation into current ABAC models (namely HGABAC), while Chapter 6 takes one of these strategies, User-to-User Attribute Delegation, and develops it into a useable delegation model with extensions for HGABAC and the HGAA architecture. The delegation strategies introduced in Chapter 4 have also been used by others (reviewed in Chapter 7 Section 7.1.3) to create ABAC delegation models following strategies other than User-to-User Attribute Delegation.

The problems of Auditability (Section 2.4.4), Security Analysis (Section 2.4.10), and Separation of Duties (Section 2.4.5) are not directly addressed by this thesis, but in some cases, are left for future work (described in Chapter 7 Section 7.3).

Bibliography

Ali E Abdallah and Etienne J Khayat. A formal model for parameterized role-based access control. In *Formal Aspects in Security and Trust*, pages 233–246. Springer, 2005.

Nabil R Adam, Vijayalakshmi Atluri, Elisa Bertino, and Elena Ferrari. A content-based authorization model for digital libraries. *IEEE Transactions on Knowledge and Data Engineering*, 14(2):296–315, 2002.

Mohammad A Al-Kahtani and Ravi Sandhu. A model for attribute-based user-role assignment. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 353–362. IEEE, 2002.

Hadiseh Seyyed Alipour and Mehdi Sabbari. Definition of action and attribute based access control rules for web services. In *Proceedings of the 2012 International Conference on Industrial Engineering and Operations Management*, pages 869–878, Istanbul, Turkey, July 2012.

Claudio Agostino Ardagna, Sabrina De Capitani di Vimercati, Gregory Neven, Stefano Paraboschi, F-S Preiss, Pierangela Samarati, and Mario Verdicchio. Enabling privacy-preserving credential-based access control with XACML and SAML. In *2010 IEEE 10th International Conference on Computer and Information Technology (CIT)*, pages 1090–1095. IEEE, June 2010.

Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, DFKI Deutsches Forschungszentrum für Künstliche Intelligenz, April 1991.

Ezedin Barka and Ravi Sandhu. Framework for role-based delegation models. In *16th Annual Conference on Computer Security Applications (ACSAC'00)*, pages 168–176. IEEE, 2000a.

Ezedin Barka and Ravi Sandhu. A role-based delegation model and some extensions. In *23rd National Information Systems Security Conference*, pages 396–404, 2000b.

Steve Barker. The next 700 access control models or a unifying meta-model? In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies*, pages 187–196. ACM, 2009.

Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James BD Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM Transactions on Information and System Security (TISSEC)*, 8(2):187–227, 2005.

Khalid Zaman Bijon, Ram Krishman, and Ravi Sandhu. Constraints specification in attribute based access control. *Science*, 2(3):pp–131, 2013.

David FC Brewer and Michael J Nash. The chinese wall security policy. In *1989 IEEE Symposium on Security and Privacy*, pages 206–214. IEEE, May 1989.

Jery Bryans. Reasoning about XACML policies using CSP. In *Proceedings of the 2005 Workshop on Secure Web Services*, pages 28–35. ACM, 2005.

Daniel J Buehrer and Chun-Yao Wang. CA-ABAC: Class algebra attribute-based access control. In *Proceedings of the 2012 IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technology-Volume 03*, pages 220–225. IEEE Computer Society, 2012.

Daniel J Buehrer, Lo Tse-Wen, and Hsieh Chih-Ming. Abia cadabia: A distributed, intelligent database architecture. *Intelligent Multimedia, Computing, and Communications*, pages 1–3, 2001.

Mike Burmester, Emmanouil Magkos, and Vassilis Chrissikopoulos. T-ABAC: An attribute-based access control model for real-time availability in highly dynamic systems. In *2013 IEEE Symposium on Computers and Communications (ISCC)*, pages 000143–000148. IEEE, July 2013.

Jan Camenisch, Sebastian Mödersheim, Gregory Neven, Franz-Stefan Preiss, and Dieter Sommer. A card requirements language enabling privacy-preserving access control. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, pages 119–128. ACM, 2010.

David W Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with X.509 attribute certificates. *Internet Computing, IEEE*, 7(2):62–69, 2003.

Yanzhe Che, Qiang Yang, Chunming Wu, and Lianhang Ma. BABAC: An access control framework for network virtualization using user behaviors and attributes. In *Proceedings of the 2010 IEEE/ACM International Conference on Green Computing and Communications & International Conference on Cyber, Physical and Social Computing*, pages 747–754. IEEE Computer Society, Dec 2010.

Yuan Cheng, Jaehong Park, and Ravi Sandhu. A user-to-user relationship-based access control model for online social networks. In *Data and Applications Security and Privacy XXVI*, pages 8–24. Springer, 2012.

Yuan Cheng, Jaehong Park, and Ravi Sandhu. Attribute-aware relationship-based access control for online social networks. In *Data and Applications Security and Privacy XXVIII*, pages 292–306. Springer, 2014.

Lorenzo Cirio, Isabel F Cruz, and Roberto Tamassia. A role and attribute based access control system using semantic web technologies. In *Proceedings of the 2007 OTM Confederated International Conference on On the Move to Meaningful Internet Systems - Volume Part II*, OTM'07, pages 1256–1266. Springer, 2007.

James Clark, Steve DeRose, and Others. XML path language (XPath). *W3C Recommendation*, 16, 1999.

Michael J Covington and Manoj R Sastry. A contextual attribute-based access control model. In *Proceedings of the 2006 International Conference on On the Move to Meaningful Internet Systems: AWeSOMe, CAMS, COMINF, IS, PKSinBIT, MIOS-CIAO, MONET - Volume Part II*, OTM'06, pages 1996–2006. Springer, 2006.

Isabel F Cruz, Rigel Gjomemo, Benjamin Lin, and Mirko Orsini. A location aware role and attribute based access control system. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 84. ACM, 2008.

Isabel F Cruz, Rigel Gjomemo, Benjamin Lin, and Mirko Orsini. A constraint and attribute based security framework for dynamic role assignment in collaborative environments. In *Collaborative Computing: Networking, Applications and Worksharing*, pages 322–339. Springer, 2009.

Ni Dan, Shi Hua-Ji, Chen Yuan, and Guo Jia-Hu. Attribute based access control (ABAC)-based cross-domain access control in service-oriented architecture (SOA). In *2012 International Conference on Computer Science & Service System (CSSS)*, pages 1405–1408. IEEE, Aug 2012.

Agostino Dovier, Carla Piazza, Enrico Pontelli, and Gianfranco Rossi. Sets and constraint logic programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 22 (5):861–931, 2000.

Ali Esmaeeli and Hamid Reza Shahriari. Privacy protection of grid service requesters through distributed attribute based access control model. In *Proceedings of the 5th International Conference on Advances in Grid and Pervasive Computing*, GPC'10, pages 573–582. Springer, 2010.

S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281, RFC Editor, April 2002. URL https://www.ietf.org/rfc/rfc3281.txt.

S. Farrell, R. Housley, and S. Turner. An Internet Attribute Certificate Profile for Authorization. RFC 5755, RFC Editor, January 2010. URL https://tools.ietf.org/html/rfc5755.

David Ferraiolo. Towards an ABAC family of models. https://csrc.nist.gov/CSRC/media/Projects/Attribute-Based-Access-Control/documents/july2013_workshop/july2013_abac_workshop_abac-model-framework_dferraiolo.pdf, July 2013. Accessed: 2020-03-17.

David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The policy machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

David Ferraiolo, Serban Gavrila, and Wayne Jansen. Policy machine: Features, architecture, and specification. Technical Report NISTIR 7987 Revision 1, National Institute of Standards and Technology, October 2015.

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.

Elena Ferrari, Nabil R Adam, Vijayalakshmi Atluri, Elisa Bertino, and Ugo Capuozzo. An authorization system for digital libraries. *The VLDB Journal*, 11(1):58–67, 2002.

Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd Millstein. Fine-grained access control with object-sensitive roles. In *Proceedings of the 23rd European Conference of ECOOP 2009 — Object-Oriented Programming*, pages 173–194. Springer, 2009.

Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering*, pages 196–205. ACM, 2005.

Mei Ge and Sylvia L Osborn. A design for parameterized roles. In *Research Directions in Data and Applications Security XVIII*, pages 251–264. Springer, 2004.

Luigi Giuri and Pietro Iglio. Role templates for content-based access control. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 153–159. ACM, 1997.

Ruo-Fei Han, Hou-Xiang Wang, Qian Xiao, Xiao-Pei Jing, and Hui Li. A united access control model for systems in collaborative commerce. *Journal of Networks*, 4(4):279–289, 2009.

Zhengqiu He, Lifa Wu, Huabo Li, Haiguang Lai, and Zheng Hong. Semantics-based access control approach for web service. *Journal of Computers*, 6(6):1152–1161, 2011.

Richard Dean Holowczak. *Extractors for Digital Library Objects*. PhD thesis, Rutgers University, Department of MS/CIS, 1997.

Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member Submission*, 21:79, 2004.

Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 800:162, 2013.

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. ACM, 2012.

INCITS. Information technology - next generation access control - functional architecture (NGAC-FA). Technical Report INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, March 2013.

INCITS. Information technology - next generation access control – generic operations and data structures (NGAC-GOADS). Technical Report INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, August 2015.

Peng Jin and Yang Fang-chun. Description logic modeling of temporal attribute-based access control. In *2006 First International Conference on Communications and Electronics*, pages 414–418. IEEE, Oct 2006.

Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012a.

Xin Jin, Ravi Sandhu, and Ram Krishnan. RABAC: Role-centric attribute-based access control. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security: Computer Network Security*, MMM-ACNS'12, pages 84–96. Springer, 2012b.

James BD Joshi, Elisa Bertino, Usman Latif, and Arif Ghafoor. A generalized temporal role-based access control model. *IEEE Transactions on Knowledge and Data Engineering*, 17 (1):4–23, 2005.

Florian Kerschbaum. An access control model for mobile physical objects. In *Proceedings of the 15th ACM Symposium on Access Control Models and Technologies*, pages 193–202. ACM, 2010.

Etienne J Khayat and Ali E Abdallah. A formal model for flat role-based access control. In *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'03)*, volume 4, 2003.

Vladimir Kolovski, James Hendler, and Bijan Parsia. Analyzing web access control policies. In *Proceedings of the 16th International Conference on World Wide Web*, pages 677–686. ACM, 2007.

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

Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. Attribute based access control for grid computing. 2006. URL http://www.mcs.anl.gov/uploads/cels/papers/P1367.pdf.

Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2): 169–180, 2009.

Bo Lang, Hangyu Li, and Wenting Ni. Attribute-based access control for layered grid resources. In *Communication and Networking*, pages 31–40. Springer, Berlin, Heidelberg, 2010.

Adam J Lee and Marianne Winslett. Open problems for usable and secure open systems. In *Workshop on Usability Research Challenges for Cyberinfrastructure and Tools Held in Conjunction With ACM CHI*, 2006.

Jaewon Lee, Heeyoul Kim, and Joon Sung Hong. An attribute aggregation architecture with trust-based evaluation for access control. In *NOMS 2008-2008 IEEE Network Operations and Management Symposium*, pages 1011–1014, April 2008.

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

Ninghui Li and William H Winsborough. Beyond proof-of-compliance: Safety and availability analysis in trust management. In *2003 Symposium on Security and Privacy*, pages 123–139. IEEE, May 2003.

Feng Liang, Haoming Guo, Shengwei Yi, and Shilong Ma. A multiple-policy supported attribute-based access control architecture within large-scale device collaboration systems. *Journal of Networks*, 7(3):524–531, 2012.

Dan Lin, Prathima Rao, Elisa Bertino, Ninghui Li, and Jorge Lobo. EXAM: a comprehensive environment for the analysis of access control policies. *International Journal of Information Security*, 9(4):253–273, 2010.

Emil Lupu and Morris Sloman. Reconciling role based management and role based access control. In *Proceedings of the Second ACM Workshop on Role-Based Access Control*, pages 135–141. ACM, 1997.

Deborah L McGuinness, Frank Van Harmelen, and Others. OWL web ontology language overview. *W3C Recommendation*, 2004.

Matunda Nyanchama and Sylvia L Osborn. The role graph model and conflict of interest. *ACM Transactions on Information and System Security (TISSEC)*, 2(1):3–33, 1999.

Jaehong Park and Ravi Sandhu. The UCON ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, 2004.

Eric Prud'Hommeaux, Andy Seaborne, and Others. SPARQL query language for RDF. *W3C Recommendation*, 15, 2008.

Carlos E Rubio-Medrano, Clinton D'Souza, and Gail-Joon Ahn. Supporting secure collaborations with attribute-based access control. In *2013 9th International Conference Conference on Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom)*, pages 525–530. IEEE, Oct 2013.

Amit Sasturkar, Ping Yang, Scott D Stoller, and CR Ramakrishnan. Policy analysis for administrative role based access control. In *19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 13–pp. IEEE, 2006.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *7th International Symposium on Foundations and Practice of Security (FPS'2014)*, pages 187–204. Springer, Nov. 2014.

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017.

Daniel Servos, Sabah Mohammed, Jinan Fiaidhi, and Tai hoon Kim. Extensions to ciphertext-policy attribute-based encryption to support distributed environments. *International Journal of Computer Applications in Technology*, 47(2):215–226, 2013.

Basit Shafiq, Elisa Bertino, and Arif Ghafoor. Access control management in a distributed environment supporting dynamic collaboration. In *Proceedings of the 2005 Workshop on Digital Identity Management*, pages 104–112. ACM, 2005.

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, Dec 2006.

Haibo Shen. A semantic-aware attribute-based access control model for web services. In *Proceedings of the 9th International Conference on Algorithms and Architectures for Parallel Processing*, ICA3PP '09, pages 693–703. Springer, 2009.

Waleed W Smari, Jian Zhu, and Patrice Clemente. Trust and privacy in attribute based access control for collaboration environments. In *Proceedings of the 11th International Conference on Information Integration and Web-based Applications & Services*, pages 49–55. ACM, 2009.

Waleed W Smari, Patrice Clemente, and Jean-Francois Lalande. An extended attribute based access control model with trust and privacy: Application to a collaborative crisis management system. *Future Generation Computer Systems*, 31:147–168, 2014.

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. ACM, 2007.

He Wang and Sylvia L Osborn. Delegation in the role graph model. In *Proceedings of the Eleventh ACM Symposium on Access Control Models and Technologies*, pages 91–100. ACM, 2006.

He Wang and Sylvia L Osborn. Static and dynamic delegation in the role graph model. *IEEE Transactions on Knowledge and Data Engineering*, 23(10):1569–1582, 2011.

Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pages 45–55. ACM, 2004.

Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*, pages 53–70. Springer, 2011.

Yonghe Wei, Chunjing Shi, and Weiping Shao. An attribute and role based access control model for service-oriented environment. In *2010 Chinese Control and Decision Conference*, pages 4451–4455. IEEE, May 2010.

Alma Whitten and J Doug Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *Usenix Security*, volume 348, 1999.

Jian Shu Lianghong Shi Bing Xia and Linlan Liu. Study on action and attribute-based access control model for web services. In *2009 Second International Symposium on Information Science and Engineering*, pages 213–216, Dec 2009.

Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies from RBAC policies. In *10th International Conference and Expo on Emerging Technologies for a Smarter World (CEWIT)*, pages 1–6. IEEE, Oct 2013.

Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies from logs. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 276–291. Springer, 2014.

Zhongyuan Xu and Scott D Stoller. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 12(5):533–545, 2015.

Danfeng Yao, Michael Shin, Roberto Tamassia, and William H Winsborough. Visualization of automated trust negotiation. In *IEEE Workshop on Visualization for Computer Security (VizSEC 05)*, pages 65–74. IEEE, Oct 2005.

Eric Yuan and Jin Tong. Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services (ICWS'05)*, page 569. IEEE, July 2005.

Guoping Zhang, Jing Liu, and Jianbo Liu. Protecting sensitive attributes in attribute based access control. In *International Conference on Service-Oriented Computing (ICSOC)*, pages 294–305. Springer, 2013.

Xinwen Zhang, Sejong Oh, and Ravi Sandhu. PBDM: a flexible delegation model in RBAC. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies*, pages 149–157. ACM, 2003.

Xinwen Zhang, Yingjiu Li, and Divya Nalla. An attribute-based access matrix model. In *Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 359–363. ACM, 2005.

Yongsheng S Zhang, Mingfeng F Wu, Lei Wu, and Yuanyuan Y Li. Attribute-based access control security model in service-oriented computing. In *Proceedings of the 2012 International Conference on Cybernetics and Informatics*, pages 1473–1479. Springer, 2014.

Jian Zhu and Waleed W Smari. Attribute based access control and security for collaboration environments. In *2008 IEEE National Aerospace and Electronics Conference*, pages 31–35. IEEE, July 2008.

Yiqun Zhu, Jianhua Li, and Quanhai Zhang. General attribute based RBAC model for web services. *Wuhan University Journal of Natural Sciences*, 13(1):81–86, 2008.

Chapter 3

HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control

3.1 Introduction

Until recently, access control research and real world access control implementations have largely fallen under one of the three traditional models of access control: Discretionary Access Control (DAC) [Lampson 1974], Mandatory Access Control (MAC) [Bell and Padula 1974; Denning 1976] or Role-Based Access Control (RBAC) [Ferraiolo et al. 2001; Sandhu et al. 1996]. In these models, access control decisions are largely based on the identity of the user. In DAC this often takes the form of an Access Control List (ACL) mapping users to permissions on an object, while MAC is based around a security lattice controlling the direction of information flow. In dynamic environments where information sharing between systems and users from different security domains is common, these identity-based access control models are inadequate. While RBAC provides a more generalized model than MAC or DAC [Osborn et al. 2000], it also falls short in cases where users and their respective roles in the system are poorly defined beforehand. A secondary issue, common among these models, is the simplicity of the access control policies. In the case of RBAC, all access control policies must fit the form of “if a user is assigned a role X they are granted the set of permissions Y”. However, this is insufficiently flexible for many real world scenarios. For example, a bank may only permit an employee with the role “teller” to access clients’ accounts during set times or limit their access to accounts based on a system wide threat level. In both cases, the policy would be too complex to express in a traditional RBAC model.

To date, researchers have largely approached this problem by extending foundational RBAC models to compensate for inadequate flexibility required for their particular use case (e.g. [Chandran and Joshi 2005; Kuhn et al. 2010; Chen and Crampton 2011]). However, there has been a growing demand from both government and industry for a more general and dynamic model of access control, namely Attribute-Based Access Control (ABAC). Rather than basing access control decisions on a user’s identity like the traditional methods, ABAC bases access control around the attributes of a user, the objects being accessed, the environment and a number of other attribute sources. Ideally, these are all properties of the elements already existing in the system and do not need to be manually entered by administration (e.g. many of the attributes about a document come from its existing metadata; author, title, etc.). Access policies can be created, limiting access to certain resources or objects, based on the result of a boolean statement comparing attributes, for example “`user.age >= 18 OR object.owner == user.id`”. This allows for flexible enforcement of real world policies, while only requiring knowledge of some subset of attributes about a given user.

Despite the demand for and potential advantages of ABAC, little has been accomplished in the way of formalizing foundational models and large scale adoption is still in its early stages (an open problem for ABAC research as discussed in Chapter 2 Section 2.4.1). The work detailed in this chapter seeks to provide a formalized hierarchical model of ABAC, entitled Hierarchical Group and Attribute-Based Access Control (HGABAC), which introduces a group based hierarchical representation of object and user attributes that is lacking in current models (a problem identified in Chapter 2 Section 2.4.3). HGABAC is intended to be a starting point that is detailed enough for real world use but generic enough to emulate traditional models of access control (another open problem, Chapter 2 Section 2.4.2).

The rest of this chapter is organized as follows. Section 3.2 reviews the existing work related to attribute-based access control models and current efforts towards standardization. Section 3.3 outlines our proposed model of ABAC, HGABAC, and provides a formal specification, as well as details of the policy language used (Section 3.3.2) and the group hierarchy (Section 3.3.1.6). Section 3.4 gives an example use case and evaluates the solution HGABAC provides. Section 3.5 demonstrates how HGABAC can be configured to emulate DAC, MAC and RBAC access control policies. Finally, Section 3.6 details our conclusions and plans for future work.

3.2 Related Work

One of the most frequently referenced works in the ABAC literature is the eXtensible Access Control Markup Language (XACML) standard [Godik and Moses 2002]. XACML is an XML-based access control policy language that is notable for its support of attribute-based policies and use in multiple access control products. While XACML supports attribute-base access control concepts and hierarchical resources, it intently lacks any kind of formal model (focus-

Table 3.1: Comparison of HGABAC with notable models of attribute-based access control.

	Logic-based Framework for ABAC [Wang et al. 2004]	**ABAC$_\alpha$** [Jin et al. 2012]	**ABAC for Web Services** [Yuan and Tong 2005]	**WS-ABAC** [Shen and Hong 2006]	**ABMAC** [Lang et al. 2009]	**HGABAC**
Hierarchical	Hierarchical attributes, no user groups					✓
Object Attributes		✓	✓	✓	✓	✓
User Attributes	✓	✓	✓	✓	✓	✓
Environment Attributes			✓	✓	✓	✓
Connection Attributes					Shown in example but not model	✓
Administrative Attributes						✓
General Model	✓	✓	For web services	For web services	For grid computing	✓
Formal Model	Only models policies and evaluation	✓	Simplistic	Simplistic	✓	✓
Can Model DAC, MAC, and RBAC	Not demonstrated	✓	Not demonstrated	Not demonstrated	Not demonstrated	✓

ing primarily on being a policy language) and instead relies on the implementing system to apply an underlying model of access control. Another related but distinct research area from ABAC is Attribute-Based Encryption (ABE), where objects are encrypted based on attribute related access policies. In Ciphertext-Policy Attribute-Based Encryption (CP-ABE) style ABE an attribute-based policy is used to encrypt an object and a user's key consist of a set of attributes describing that user [Bethencourt et al. 2007; Servos et al. 2013]. While ABE, much like XACML, lacks any kind of formal ABAC model and has rather simplified access policies, it does provide an interesting means of enforcing ABAC policies outside of the security domain in which they originate.

Various works have attempted to informally describe ABAC or have taken the first steps towards formalization. The most notable of these are summarized and compared to our model in Table 3.1. Yuan and Tong [Yuan and Tong 2005] describe the ABAC model in terms of authorization architecture and policy engineering and give an informal comparison between ABAC and traditional role-based models. Shen and Hong [Shen and Hong 2006] present WS-ABAC, an ABAC model designed for web services based around XACML. However, the model presented in this work is limited and mostly describes an architecture to use XACML and attribute-based policies to provide authentication for web services. Lang et al.'s ABMAC model[Lang et al. 2009] aims to bring ABAC like access control to grid computing. While this model is based on attribute-based policy decisions, it has several key differences in the policy description and policy evaluation methods. This work is of note as it mentions ABAC's ability to represent the traditional models using a single policy language.

Wang et al.[Wang et al. 2004] propose a logic-based framework for ABAC based on logic programming where policies are specified as "stratified constraint flounder-free logic programs that admit primitive recursion". While their framework introduces hierarchical attributes (something lacking from other models), it is largely focussed on the representation, consistency and performance of attribute-based policies and their evaluation over providing a workable model of ABAC. Several critical components required for a usable model are absent, including object attributes (the only attributes considered are user attributes) and they omit any kind of formalization of ABAC aspects outside of policies and their evaluation (e.g. there is no mention of objects and only access control on services/operations is considered).

Lastly, and most promising is the work by Jin et al. [Jin et al. 2012] towards a generalized and formalized model of ABAC with constraints for the traditional models, which they call $ABAC_\alpha$. Their model provides a first step *"to develop a formal ABAC model that is just sufficiently expressive to capture DAC, MAC and RBAC"* which allows configuration of constraints on attributes at creation and modification time as well as policies. While this work provides a sufficient basis for new models of ABAC, it (intentionally) lacks components that would be necessary for a real world implementation, such as attribute and object hierarchies, a user friendly policy language and environment attributes.

Our model, HGABAC, is distinct from other models in several regards. Most notably, the graph based user group hierarchy provides several new interesting means of allowing policy administrators to represent the traditional models in an ABAC framework (allowing the hierarchy to model MAC and RBAC in an intuitive way, rather than a partially ordered set as is done in $ABAC_\alpha$ [Jin et al. 2012]). These hierarchical representations of MAC and RBAC are demonstrated in Section 3.5. The object group hierarchy allows for objects to be categorized into collections of similar types of objects (e.g. a collection of only health care record objects) and have common attributes applied to all members of the group. This reduces the amount of manual intervention required to tag similar objects with a common attribute and value pairs and reduces the number of object attribute assignments required (as evaluated in Section 3.4.2). Additionally, several efforts are made to create a model more suited to real world application without losing descriptive power or flexibility in terms of policies that may be enforced; a fully specified and intuitive policy language is presented (in Section 3.3.2) loosely based on C style boolean statements and a more rich selection of attribute sources is allowed. Attributes based on the user's current connection to the system and administrative attributes are supported which are lacking in other models. Finally, HGABAC uses a strongly typed system to represent attribute values (i.e. an attribute must have a predefined data type, e.g. integer, floating point, set, etc.). This helps enforce consistency in policies and attribute value assignments as well as helping to prevent any possible ambiguity in policies (e.g. preventing any type mismatches).

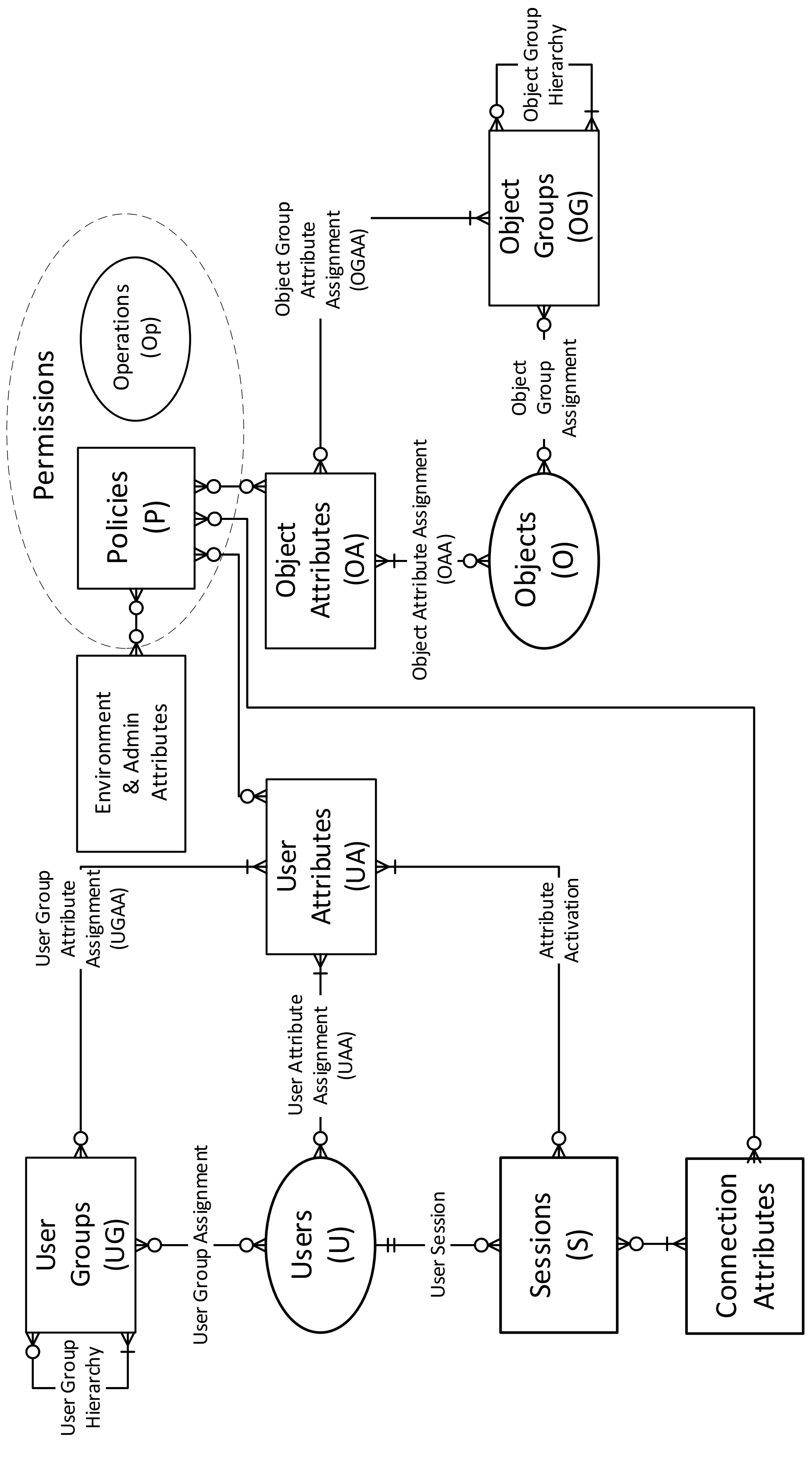


Figure 3.1: HGABAC components and relations using Crow's Foot Notation to denote cardinality of relationships. Primitive components are shown in ovals.

3.3 HGABAC Model

3.3.1 Formal Model

3.3.1.1 Basic Elements and Definitions

We define the base elements of the HGABAC model, as shown in Figure 3.1, as follows:

- **Users (U):** set of current human and non-human entities that may request access on system resources through sessions.
- **Objects (O):** set of system resources (files, database records, devices, etc.) for which access should be limited.
- **Operations (Op):** set of all operations provided by the system that may be applied to an object (e.g. read, write, create, delete, update etc.).
- **Policies (P):** set of all current policy strings following the format of our policy language defined in Section 3.3.2.
- **Sessions (S):** set of all user sessions, such that each element, s is a tuple of the form $s = (u \in U, a \subseteq effective(u \in U), con\_atts)$ where u is the user who activated the session, con_atts is the set of connection attributes for the session such that $\forall c \in con\_atts : c = (name, values)$ and a is the subset of the user's effective attributes they wish to activate for the given session ($effective(u)$ is the set of all attributes a user is assigned either directly through the UAA relation or indirectly through group membership). Policies are evaluated on the basis of the activated attributes in a user's session rather than the total set of the user's assigned and inherited attributes.
- **Permissions:** pairing of a policy string and a set of operations, such that $perm = (p \in P, op \subset Op)$. Access to perform a set of operations, op, on a given object is only allowed if there exists a permission that contains a policy, p, that is satisfied by a given set of attributes corresponding to the requesting user's session, object being accessed and the current state of the connection, environment and administrative attributes in the system. For example, a policy paired with a read operation, "`user.id = object.author`", would allow read access to all objects for which the user is also the author.

3.3.1.2 Attributes

HGABAC defines attributes to be (name, value, type) triples where the name is a unique identifier and value is an unordered set of atomic values of a given type or the *null* set. Type restricts the data type of the atomic values (e.g. string, integer, boolean, etc.) to a system defined data type. Attributes represent some descriptive characteristic of the entity to which they are assigned. For example, a user might have attributes describing their name, age, employee id, etc., while an object might have attributes describing its author, owner, file type, etc. The set of all attributes (TA) is divided into five subsets based on their origin and to which entity or object they may be applied:

- **User Attributes (UA):** the set of attribute name, type pairs that may be applied to users such that $\forall a \in UA : a = (name, type)$ and each element of UA has a globally unique name (i.e. there cannot be two elements with the same name but different types). Note that value is left out of the definition of UA as user attributes are given a set of values when assigned to users directly or to groups (in the UAA and UGAA relations).

- **Object Attributes (OA):** the set of attribute name, type pairs that may be applied to objects such that $\forall a \in OA : a = (name, type)$ and each element of OA has a globally unique name (i.e. there cannot be two elements with the same name but different types). As with UA, value is left out of the definition of OA as object attributes are given a set of values when assigned to objects directly or to groups (in the OAA and OGAA relations).

- **Environment Attributes (EA):** the set of attribute (name, value, type) triples that represent the current state of the system's environment (e.g. the current time, number of active users, etc.) such that $\forall a \in EA : a = (name, value, type)$ and each element of EA has a globally unique name (i.e. there cannot be two elements with the same name but different types or values). What properties of a system's environment are available as environment attributes is left to the implementation.

- **Connection Attributes (CA):** the set of attribute name, type pairs that correspond to attributes derived from and available for each connection to the system such that $\forall a \in CA : a = (name, type)$ and each element of CA has a globally unique name (i.e. there cannot be two elements with the same name but different types). What properties of the connection are available as connection attributes is left as a implementation decision; however, at a minimum some kind of unique session id should be included.

- **Administrative Attributes (AA):** the set of attribute (name, value, type) triples that are defined by administrators (including automated administrative tasks and programs) that rarely change and apply to all policies which reference them such that $\forall a \in AA : a = (name, value, type)$ and each element of AA has a globally unique name. What administrative attributes are available will change at runtime based on both the implementation and actions of administrators.

- **Total Attributes (TA):** set of all attributes that exist in a given system such that $TA = UA \cup OA \cup CA \cup EA \cup AA$.

3.3.1.3 Groups

Groups and their hierarchies both simplify administration tasks, allowing system administrators to assign attributes to groups of users or objects at once rather than directly, and allow for more intuitive and expressive configuration possibilities than current ABAC models (including in the task of emulating the traditional models as shown in Section 3.5). Groups, their hierarchy, and memberships are created by system administrators in conjunction with and at the same time as policy creation. Section 3.3.1.6 details the group hierarchy, while user and object group definitions and membership are defined below:

- **User Groups (UG):** set of all current user groups, where each element is comprised of a tuple, g, such that $g = (name, u \subseteq U, p \subseteq UG)$ where *name* is a globally unique identifier, u is the set of members of the group, and p is the set of the group's parents in the user group graph.

- **Object Groups (OG):** set of all current object groups, where each element is comprised of a tuple, g, such that $g = (name, o \subseteq O, p \subseteq OG)$ where *name* is a globally unique identifier, o is the set of members of the group, and p is the set of the group's parents in the object group graph.

3.3.1.4 Relations

We define the following relations between the base elements, groups and attributes:

- **Direct User Attribute Assignment (UAA):** user attribute assignment relation containing user, attribute name, value triples such that:

$$\forall uaa \in UAA : uaa = (u \in U, att\_name, values) \tag{3.1}$$

where $att\_name \in \{name \mid (name, type) \in UA\}$ and $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in UA$. There may exist only one tuple in UAA for every user, att_name pair.

- **Direct Object Attribute Assignment (OAA):** object attribute assignment relation containing object, attribute name, value triples such that:

$$\forall oaa \in OAA : oaa = (o \in O, att\_name, values) \tag{3.2}$$

where $att\_name \in \{name \mid (name, type) \in OA\}$ and $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in OA$. There may exist only one tuple in OAA for every object, att_name pair.

- **User Group Attribute Assignment (UGAA):** user group attribute assignment relation containing user group name, attribute name, value triples such that:

$$\forall ugaa \in UGAA : ugaa = (group\_name, att\_name, values) \tag{3.3}$$

where $group\_name \in \{name \mid (name, u, p) \in UG\}$ and $att\_name \in \{name \mid (name, type) \in UA\}$. $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in UA$. There may exist only one tuple in UGAA for every group_name, att_name pair.

- **Object Group Attribute Assignment (OGAA):** object group attribute assignment relation containing object group name, attribute name, value triples such that:

$$\forall ogaa \in OGAA : ogaa = (group\_name, att\_name, values) \tag{3.4}$$

where $group\_name \in \{name \mid (name, u, p) \in OG\}$ and $att\_name \in \{name \mid (name, type) \in OA\}$. $values$ is some set of elements such that each element of $values$ is of the same data type ($type$) and $(att\_name, type) \in OA$. There may exist only one tuple in OGAA for every group_name, att_name pair.

3.3.1.5 Mappings

The following are the most important formal functions in the HGABAC model.

- **direct:** Mapping of a user, object, or group to the attribute name, value pairs directly assigned to it in the UAA, OAA, UGAA or OGAA relation (i.e. not including inherited attributes or attributes from group membership). *direct* is defined as:

$$direct(x) = \begin{cases} \{(n, v) \mid (x, n, v) \in UAA\}, & \text{if } x \in U \\ \{(n, v) \mid (x, n, v) \in OAA\}, & \text{if } x \in O \\ \{(n, v) \mid (name(x), n, v) \in UGAA\}, & \text{if } x \in UG \\ \{(n, v) \mid (name(x), n, v) \in OGAA\}, & \text{if } x \in OG \end{cases} \tag{3.5}$$

  where $name(x)$ is the name of the given group, n is an attribute name, v is a set of valid values for that attribute and $x \in U \cup O \cup UG \cup OG$.

- **consolidate:** Mapping of a set of attribute name, value pairs which may contain multiple instances of the same name to a set of attribute name, value pairs where each name occurs only once. Value sets are unioned together for pairs with the same attribute name. *consolidate* is defined as:

$$consolidate(x) = \{(n, v_1 \cup v_2) \mid (n, v_1) \in x \wedge (n, v_2) \in x\} \tag{3.6}$$

  where x is sets of attribute name, value pairs, n is an attribute name, v_1 and v_2 are sets of values.

- **member:** Mapping of a User or Object to the set of groups for which they are a member. *member* is defined as:

$$member(x) = \begin{cases} \{(n, u, p) \mid (n, u, p) \in UG \wedge x \in u\}, & \text{if } x \in U \\ \{(n, o, p) \mid (n, o, p) \in OG \wedge x \in o\}, & \text{if } x \in O \end{cases} \tag{3.7}$$

  where n is the name of a group, u is a subset of U, o is a subset of O, p is a subset of UG or OG and $x \in U \cup O$.

- **inherited:** Mapping of a user, object or group to its set of inherited attributes (i.e. the set of attributes assigned indirectly through the group hierarchy or group membership). *inherited* is defined as:

$$inherited(x) = consolidate(\\ \begin{cases} \{(n,v) \mid g \in member(x) \land (n,v) \in consolidate(direct(g) \cup inherited(g))\}, & \text{if } x \in U \cup O \\ \{(n,v) \mid g \in parents(x) \land (n,v) \in consolidate(direct(g) \cup inherited(g))\}, & \text{if } x \in UG \cup OG \\ \emptyset, & \text{if } name(x) = min\_group \end{cases} \\ ) \tag{3.8}$$

where $parents(x)$ is the set of parents for the given group, n is an attribute name, v is a set of valid values for n and $x \in O \cup U \cup UG \cup OG$.

- **effective:** Mapping of a user, object, or group to their effective attributes (i.e. all attributes inherited or directly assigned). $effective$ is defined as:

$$effective(x) = consolidate(direct(x) \cup inherited(x)) \tag{3.9}$$

where x is a user, object or group (i.e. $x \in U \cup O \cup OG \cup UG$).

- **name:** Mapping of a group or attribute to its assigned name. *name* is defined as:

$$name(x) = x_{(1)} \tag{3.10}$$

that is, the name is the first element of the tuple in both the case of groups and attributes, and $x \in OG \cup UG \cup TA$.

- **parents:** Mapping of a group to its set of parents. $parents$ is defined as:

$$parents(x) = x_{(3)} \tag{3.11}$$

that is, the set of a groups where parents is the third element of the tuple in both the case of user and object groups, and $x \in OG \cup UG$.

- **authorized:** $P, S, O \rightarrow \{true, false, undef\}$

Function which determines if a user session passes the given policy given the current value of the environment and administrative attributes for a given object, where P is the set of all policies, S is the set of all sessions and O is the set of all objects. $true$ and $false$ are returned as expected based on the evaluation of the boolean policy rule; $undef$ is returned if the policy cannot be evaluated (e.g. an object or user attribute referred to in the policy is not present in the attribute sets or incompatible types are compared).

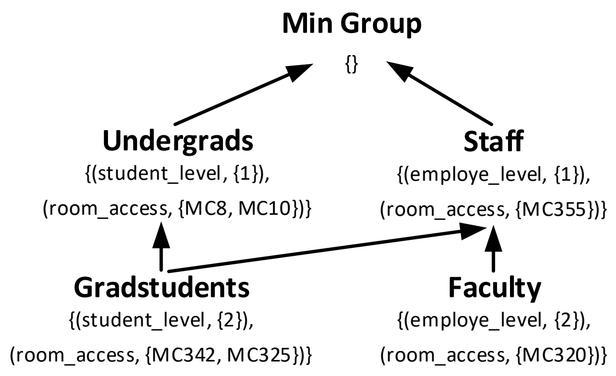


Figure 3.2: Example user group hierarchy represented as a graph. The large bold text denotes the group's name, beneath which the set of directly assigned attributes is shown.

3.3.1.6 Group Graph

The group graph allows administrators to assign attributes in a more natural way that matches their organization hierarchy and is reminiscent of the role hierarchy in RBAC. This enables administrators to not only assign, update or remove attributes to/from groups of users and objects at a time (rather than individually), but create parent/child relations between these groups such that child groups inherit the attributes of their parents. We show (in Section 3.4) that this results in fewer overall assignments (of both attributes and group memberships) when compared to traditional ABAC models, thus leading to fewer relations between attributes and users/objects/groups for administrators to manage.

HGABAC represents the group hierarchy as a directed acyclic graph with each group a vertex and each edge a parent/child relation between the groups such that the edge is directed to the parent. Additionally, all paths in the graph must eventually end at a special *min_group* that has no parents and no assigned attributes. A group, g, can only have *min_group* as a parent if it has one and only one parent such that effective(g) = direct(g) and inherited(g) = $\emptyset$. The parent/child relation between any two related groups is defined such that group c is a child of group p iff:

$$\begin{aligned}\forall(n, v_1) \in \mathit{effective}(p) :\\ \exists! a \in \mathit{effective}(c) : a = (n, v_2) \land v_1 \subseteq v_2\end{aligned} \tag{3.12}$$

A child group must have one attribute for each effective attribute assigned to the parent group, such that the attribute has the same name and the parent's attribute's value is a subset of the child's attribute's value. Thus, the effective attributes for a group, g, are calculated as:

$$\mathit{effective}(g) = \mathit{consolidate}(\mathit{direct}(g) \cup \mathit{inherited}(g)) \tag{3.13}$$

Users' and objects' effective attributes are calculated in a similar way, consolidating the values of directly assigned and inherited attributes.

An example user group hierarchy is shown in Figure 3.2. In this example the set of effective attributes of groups *Undergrad* and *Staff* are the same as their set of direct attributes as they both inherit from *min_group*. The group *Faculty* inherits the attributes (*employe_level*, {1}) and (*room_access*, {MC355}) from the group *Staff* such that the effective attributes of *Faculty*

Table 3.2: HGPL truth table for *AND*, *OR* and *NOT* operations.

X	*Y*	*X* AND *Y*	*X* OR *Y*	NOT *X*
TRUE	*TRUE*	*TRUE*	*TRUE*	*FALSE*
TRUE	*FALSE*	*FALSE*	*TRUE*	*FALSE*
FALSE	*TRUE*	*FALSE*	*TRUE*	*TRUE*
FALSE	*FALSE*	*FALSE*	*FALSE*	*TRUE*
TRUE	*UNDEF*	*UNDEF*	*TRUE*	*FALSE*
UNDEF	*TRUE*	*UNDEF*	*TRUE*	*UNDEF*
UNDEF	*FALSE*	*FALSE*	*UNDEF*	*UNDEF*
FALSE	*UNDEF*	*FALSE*	*UNDEF*	*TRUE*
UNDEF	*UNDEF*	*UNDEF*	*UNDEF*	*UNDEF*

will be (*employe_level*, {2, 1}) and (*room_access*, {MC320, MC355}). Similarly, the group *Gradstudents* inherits attributes from both the groups *Undergrads* and *Staff* such that the set of effective attributes for *Gradstudents* is {(*employe_level*, {2, 1}), (*room_access*, {MC325, MC8, MC10, MC355})}.

The object group hierarchy has the same properties as the user group hierarchy (being a directed acyclic graph, etc.), and is set up in a similar way with a *min_group* place holder being the ancestor of all groups. In implementations, of HGABAC it is likely that both the user and object group graphs could be consolidated into a single graph with the same *min_group* and treated similarly so long as constraints are enforced so no object group may inherit from a user group and no object group may have a user attribute assigned (and vice versa).

Both the user and object group graphs are generated by system administrators as part of policy creation (in conjunction with creating HGPL policies and assigning attributes). Tools for automating group graph generation, group membership, attribute assignment and verifying their correctness for given policies is out of the scope of this work and left to future work.

3.3.2 Policy Language

In HGABAC, access control decisions are based on the evaluation of Hierarchical Group Policy Language (HGPL) policies. HGPL is a boolean rule based policy language comparing attributes and constants, purpose-built for HGABAC. Hierarchical Group Policy Language (HGPL) differs from current approaches such as $ABAC_\alpha$'s CPL[Jin et al. 2012] or XACML[Godik and Moses 2002] in that it is designed to be lightweight, intuitive and easy to use while also being efficient in terms of parsing and interpreting (an evaluation of the Hierarchical Group Policy Language (HGPL) interpreter's performance is given in Chapter 5 Section 5.5.3).

HGPL is based around C-style Boolean expressions in which the result of logical operations (`AND`, `OR`, `NOT`) on ternary values (`TRUE`, `FALSE`, `UNDEF`) are determined based on the `AND`, `OR` and `NOT` truth tables from Kleene K3 logic [Kleene 1938] as shown in Table 3.2. A policy evaluated to `UNDEF` is equivalent to `FALSE` in terms of access control decisions (i.e. access is denied). Comparison operations (<, >, etc.) result in `TRUE` or `FALSE` as expected when value types are comparable (e.g. `1 < 2` results in `TRUE`) and `UNDEF` when incomparable (e.g. `"Pizza" > 3.1415`). The following definition of the policy language is given using ABNF[Crocker and Overell 1997] syntax (an updated version of this syntax, HGPLv2, is introduced in Chapter 5):

Listing 3.1. Grammar of HGABAC's HGPLv1 Policy Language in ABNF

```
policy   =  exp [ bool_op policy ]
         /  ( policy )

exp      =  var op var
         /  [ "NOT" ] bool_var
         /  [ "NOT" ] "(" policy ")"

var      =  const / att_name

bool_var =  boolean / att_name

op       =  ">" / "<" / "=" / ">=" / "<=" / "!="
         / "IN" / "SUBSET"

bool_op  =  "AND" / "OR"

att_name =  user_att_name / object_att_name / env_att_name
         / admin_att_name / connect_att_name

user_att_name    =  "user." id

object_att_name  =  "object." id

env_att_name     =  "env." id

admin_att_name   =  "admin." id

connect_att_name =  "connect." id

atomic   =  int / float / string / "NULL"

const    =  atomic / set

boolean  =  "TRUE" / "FALSE" / "UNDEF"

set      =  "{" "}" / "{" setval "}"

setval   =  atomic / atomic "," setval

id       =  +(ALPHA / DIGIT / "_")

int      =  ["-"] ( 1-9 ) *( DIGIT ) / "0"

float    =  int "." +( DIGIT )

string   =  DQUOTE *( %x20-21 / %x23-7E ) DQUOTE
```

user_att_name and *object_att_name* correspond to attribute names in *UA* and *OA* respectively, while *env_att_name*, *admin_att_name* and *connect_att_name* correspond to attribute names in *EA*, *AA* and *CA* respectively. *string* are c-style strings limited to printable characters. Otherwise, our policy language functions like c-style boolean statements where the only variables are attributes. Semantics of indivudal operations (e.g. <, >, =, *IN*, etc.) are given in Appendix B.

Example 3.1. Example HGABAC Policies Using the HGPLv1 Policy Language

(a) `user.id IN {5, 72, 4, 6, 4} OR user.id = object.owner`

(b) `object.required_perms SUBSET user.perms AND user.age >= 18`

(c) `user.admin OR (user.role = "doctor" AND user.id != object.patient)`

Example 3.1 gives three example policies. Policy string *3.1a* would only return true when processed by the *authorized* function, if the user attribute *id* is present and has at least one value matching an element in the given set {5, 72, 4, 6, 4} or has a value that is equal to a value in *object.owner*. Note that the value of the attribute *user.id* may be a set of multiple values, which would still pass the policy so long as $\exists e \in user.id : e \in \{5, 72, 4, 6, 4\}$ or $e = object.owner$. Policy string *3.1b* would limit access to a user who is at least 18 and has the set of permissions such that the object's *required_perms* is a subset. Finally, policy string *3.1c* demonstrates a possible use case, where you desire to give doctors access to any medical record but their own (as well as allow a user with the admin attribute to access any record).

3.3.2.1 HGPL Limitations

HGPL's lightweight and easy-to-use design requires limiting the features supported by the policy language. In contrast with languages such as XACML, which aim to be as fine-grain and flexible as possible, HGPL has a number of limitations:

1. **Negative permissions:** HGPL only supports positive permissions. Each policy string may grant one or more permissions but there is currently no support for policies that remove or restrict permissions from a user if satisfied. This limitation prevents conflicts when policies are combined but limits flexibility of HGPL policies.

2. **Obligations:** XACML supports the notion of obligations, directives that state what actions must be carried out before or after an access control decision is made (approved or denied). Obligations allow for mandatory logging and a means to meet more formal access control requirements. HGPL does not currently support such directives.

3. **Operations:** Currently, HGPL does not support operations or functions to be run on the values of attributes beyond the Boolean operations stated in Listing 3.1 and Appendix B. Adding built-in mathematical, string, and more complex logical operations and functions could allow for more complex and flexible policies. For example, an *AVERAGE* or *SUM* function could allow for policies to be created based on an evaluation of a set of values (e.g. *SUM(user.expenses) <= 1000* or *AVERAGE(env.temperatures) >= 30*).

4. **Namespace:** HGPLv1 does not support namespaces for attributes, permissions, users, etc. This may cause issues when attributes may have different meanings but similar names across organizational or system boundaries. HGPLv2 (Chapter 5 Section 5.4.1) resolves this by adding a URI based namespace for each access control element.

5. **Policy Combinations:** HGPLv1 lacks any means of combining or referencing other policies. HGPLv2 (Chapter 5 Section 5.4.5) provides a partial solution in the form of policy references. Policy references allow policies to reference other policies such that policies can be combined using basic logical operations.

3.4 Examples and Evaluation

3.4.1 Example: The Library

This section outlines how HGABAC may be used to provide access control for a hypothetical university library. In the following use cases it is assumed that access control is desired on four different kinds of resources provided by the library; books, course material (textbooks, lecture notes, etc.), periodicals, and archived records.

> **Case 1:** Undergraduate students may check out any unrestricted book and any course materials for a course in which they are enrolled.
>
> **Case 2:** Graduate students may check out any unrestricted book or periodical but may only check out course materials for courses in which they are a teaching assistant or enrolled.
>
> **Case 3:** Faculty may check out any book, periodical or course material as well as any archived record from their department.
>
> **Case 4:** Staff may access any resource between the hours of 8:00 and 17:00 on weekdays.
>
> **Case 5:** Students enrolled in a computer science course may access periodicals from the university network.

3.4.1.1 Group Graphs

Figure 3.3 shows the the user and object group hierarchies that would be created by an administrator for the above example and cases.

3.4.1.2 Group Membership

Users are assigned to one of the four *user_type* groups (Undergrads, Gradstudents, Faculty or Staff) as expected (i.e. undergraduate students are members of the “Undergrads” group, graduate students to the “Gradstudents” group, etc.). Students are also assigned membership in a user group for each course they are enrolled in (e.g. if a student is enrolled in CS203 they would be a member of the user group “CS203”). Graduate students and faculty also belong to a

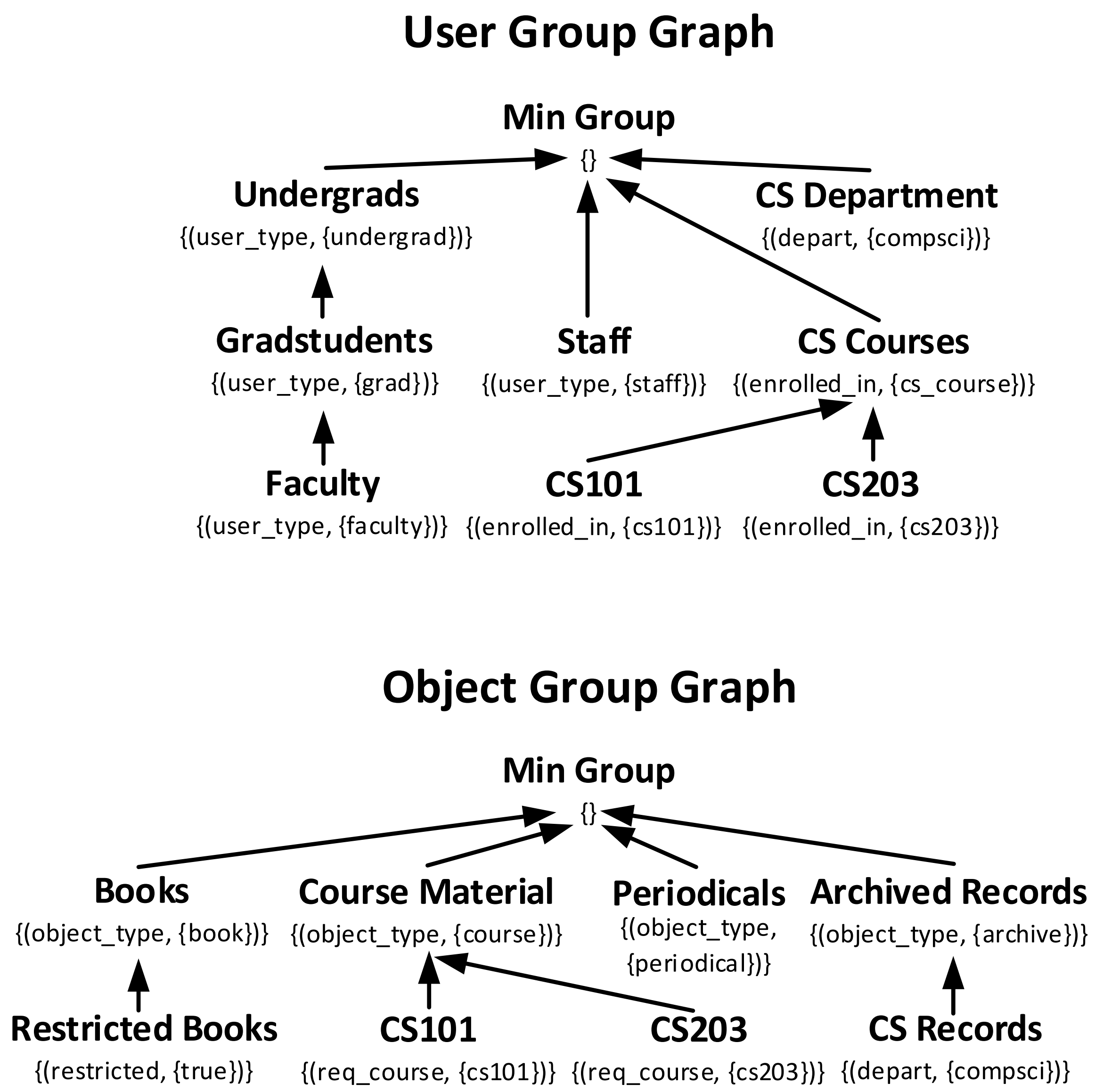


Figure 3.3: Example user and object group hierarchies to used in the cases given in Section 3.4.1.

department group (for the purposes of these cases, only a computer science department group, “CS Department”, is considered[1]). For example, a computer science graduate student taking the course CS203 would be a member of the “Gradstudents”,“CS203” and “CS Department” groups and would have the effective attributes {{*user_type,* {*undergrad, grad*}}, {*enrolled_in,* {*cs_course, cs203*}}, {*depart,* {*compsci*}}.

Resources are assigned membership in one of the four *object_type* groups or one of their children as expected (i.e. books are assigned to the “Books” group or the “Restricted Books” group, course material to one of the course object groups (e.g. “CS101”), etc.). For example a textbook for the course CS101 would be assigned to the “CS101” object group and would have the following effective attributes {{*object_type,* {*course*}}, {*req_course,* {*cs101*}}}.

3.4.1.3 Case 1

One permission pair would be sufficient for meeting the requirements of the case:

```
PERMS = {
   { ""undergrad" IN user.user_type AND (
       (object.object_type = "book" AND NOT object.restricted) OR
       (object.object_type = "course" AND
          user.enrolled_in IN object.req_course) )"
   , check_out_book }
}
```

where *check_out_book* is the operation that allows a resource to be read/viewed.

3.4.1.4 Case 2

In this case each graduate student would be assigned an attribute “teaching” containing the set of courses the graduate student is assigned to as a TA. The following permission pair combined with the pair from Case 1 would be sufficient for meeting the requirements of the case:

```
PERMS = {
   { ""grad" IN user.user_type AND (
       object.object_type = "periodical" OR (
          oject.object_type = "course" AND
             object.req_course IN user.teaching) )"
   , check_out_book }
}
```

As the “Gradstudents” group is a child of the “Undergrads” group, graduate students are granted access to unrestricted books and course materials for courses they are enrolled in through the policy permission pair in Case 1 (as they have both the values “grad” and “undergrad” for their *user_type* attribute).

[1]In a real-world system, it would be likely that a full departmental hierarchy would be present. For example, each department in the university would have a user group with their school or faculty as a parent user group. In the case of a computer science department, this parent group might be the Faculty of Science. This hierarchy would closely represent the organization hierarchy of the university.

3.4.1.5 Case 3

As this case is less restrictive than the previous it can be met by a straightforward permission pair:

```
PERMS = {
  { ""faculty" IN user.user_type AND
     ( object.object_type IN {"book", "periodical", "course"} OR
        ( object.object_type = "archive" AND
           object.depart IN user.depart ) )"
  , check_out_book }
}
```

3.4.1.6 Case 4

For this case at least two environment attributes are required. “time_of_day_hour”, that represents the current hour (1 to 24) and, “day_of_week”, that represents the current day of the week (1 to 7). Then the following permission pair would be sufficient for meeting the requirements for the case:

```
PERMS = {
  { ""staff" IN user.user_type AND
     env.time_of_day_hour >= 8 AND env.time_of_day_hour <= 16
     AND env.day_of_week IN {2, 3, 4, 5, 6}"
  , check_out_book }
}
```

3.4.1.7 Case 5

It is assumed that four connection attributes exist which represent the user’s IP address; “ip_octet_1” represents the first digit of the user’s IP address, “ip_octet_2”, the second and so on up to “ip_octet_4”. It is also assumed that all IP addresses matching the pattern “192.168.*.*” are internal addresses on the university’s network. The following permission pair would then be sufficient for meeting the requirements of the case:

```
PERMS = {
  {""cs_course" IN user.enrolled_in AND
     connect.ip_octet_1 = 192 AND
     connect.ip_octet_2 = 168 AND
     object.object_type = ‘‘periodical""
  , check_out_book }
}
```

3.4.2 Evaluation

To evaluate whether the hierarchical user and object groups of the HGABAC model provides an advantage over more traditional non hierarchical models of ABAC in terms of simplifying administration and reducing complexity, we evaluate HGABAC based on the number of attribute and group assignments needed to fulfill the requirements of each use case given in Section 3.4.1. These results are compared to the number of attribute assignments that would be required in a non hierarchical model of ABAC such as ABAC$_\alpha$ [Jin et al. 2012] (if ABAC$_\alpha$ supported environment and connection attributes required to model cases 4 and 5).

Table 3.3 outlines the results of this comparison. The worst case (each user is enrolled in each course and each object is of an *object_type* such that it will have the most attributes) is assumed as well as a constant number of courses and departments (the same number shown in the group graphs in Figure 3.3). In cases 1, 2 and 3 where it is required that multiple attributes be assigned to each object and user, HGABAC has a noticeable advantage as hierarchical groups allow multiple attributes to be assigned with a single group membership assignment. This also has significant advantages for administration of ABAC systems, for example if an administrative tasks required adding an attribute to every student in a given course, only a single additional attribute assignment to the course's user group would be required in HGABAC, while a new attribute assignment for every user in the course would be required in traditional ABAC. Cases 4, and 5 take less advantage of HGABAC's group hierarchy, instead making use of connection and environment attributes, and as such results in HGABAC having a comparable performance to traditional ABAC but with a slight overhead due to the object and user groups.

Table 3.3: Number of attribute and group assignments required for each case in Section 3.4.1. U is the number of users and O is the number of objects.

	Case 1		**Case 2**		**Case 3**	
	HGABAC	ABAC	HGABAC	ABAC	HGABAC	ABAC
User Attribute	4	$4U$	$U+5$	$5U$	4	$2U$
Object Attribute	5	$2O$	6	$2O$	8	$2O$
User Group	$3U$	0	$3U$	0	$2U$	0
Object Group	O	0	O	0	O	0
Total	$3U+O+9$	$4U+2O$	$4U+O+11$	$5U+2O$	$2U+O+12$	$2U+2O$
	Case 4		**Case 5**			
	HGABAC	ABAC	HGABAC	ABAC		
User Attribute	1	U	1	U		
Object Attribute	0	0	1	O		
User Group	U	0	U	0		
Object Group	0	0	O	0		
Total	$U+1$	U	$U+O+2$	$U+O$		

3.5 Emulating Traditional Models

3.5.1 DAC Style Configuration

HGABAC can be configured to emulate DAC by assigning each user an "id" attribute with a single value equal to a unique identifier for that user and assigning each object an attribute for each access mode (e.g. "read" and "write") that contains the set of user ids corresponding to users who have access to that object for the given access mode. The set of permissions are then simply: `PERMS = {("user.id IN object.read", read), ("user.id IN object.write", write)}`. To model DAC style administration, an "owner" attribute maybe added to objects that contains a single user id corresponding to the owner of the object. The permission to grant access on administrative operations is then simply: `("user.id = object.owner", admin_operation)`.

3.5.2 MAC Style Configuration

HGABAC's user groups allow configurations that emulate MAC style lattice based access control. For example given the following (Figure 3.4) MAC lattice:

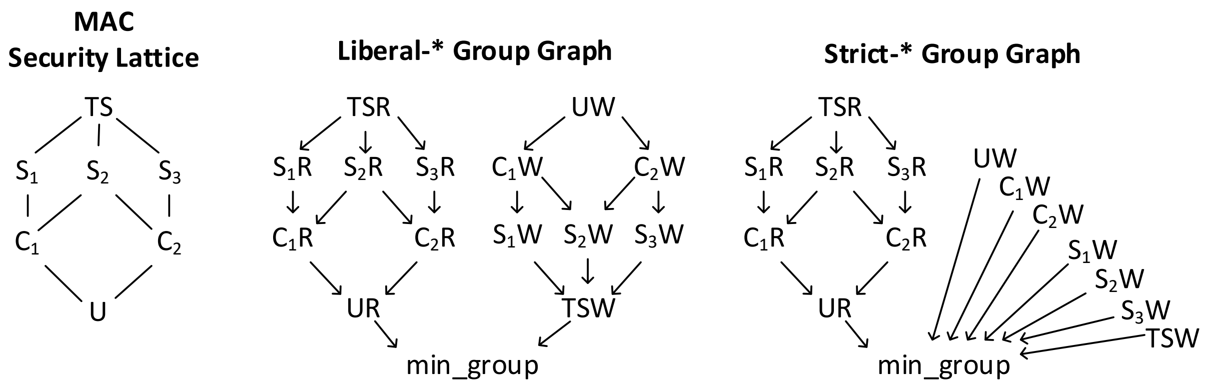


Figure 3.4: Example MAC lattice (left), required user group graph to repersent liberal-* property (middle) and strict-* property (right).

The user group graph may be configured as follows to enable MAC with a liberal *-property where each user is assigned only to a single read group and a single write group. This is similar to how RBAC is configured to emulate MAC in [Osborn et al. 2000]. Each read group is assigned a single attribute named "read" and each write group is assigned a single attribute named "write" both with a single value equal to its clearance level (e.g. group UR is assigned the value {"UR"} for its "read" attribute). Each object is assigned a security level attribute named "level". The set of permissions are then simply: `PERMS = {("object.level IN user.read", read), ("object.level IN user.write", write)}`. Users are limited to only activating attributes inherited from groups of a single security level in any given session. Table 3.4 shows `direct(g)` and `effective(g)` for each group.

Table 3.4: Resulting `direct(g)` and `effective(g)` attribute sets for each group (`g`) in the liberal-* User Group Graph given in Figure 3.4.

g	direct(g)	effective(g)
min_group	$\emptyset$	$\emptyset$
UR	"UR"	"UR"
C_1R	"C1R"	"UR", "C1R"
C_2R	"C2R"	"UR", "C2R"
S_1R	"S1R"	"UR", "C1R", "S1R"
S_2R	"S2R"	"UR", "C1R", "C2R", "S2R"
S_3R	"S3R"	"UR", "C2R", "S3R"
TSR	"TSR"	"UR", "C1R", "C2R", "S1R", "S2R", "S3R", "TSR"
TSW	"TSW"	"TSW"
S_1W	"S1W"	"TSW", "S1W"
S_2W	"S2W"	"TSW", "S2W"
S_3W	"S2W"	"TSW", "S3W"
C_1W	"C1W"	"TSW", "S1W", "S2W", "C1W"
C_2W	"C2W"	"TSW", "S2W", "S3W", "C2W"
UW	"UW"	"TSW", "S1W", "S2W", "S3W", "C1W", "C2W", "UW"

3.5.3 RBAC Style Configuration

HGABAC's user groups can also effectively enforce hierarchical RBAC style access control by having each user group represent a role and its assigned attributes, represent permissions. For example given the following role hierarchy (Figure 3.5), the user group graph on the right may be used:

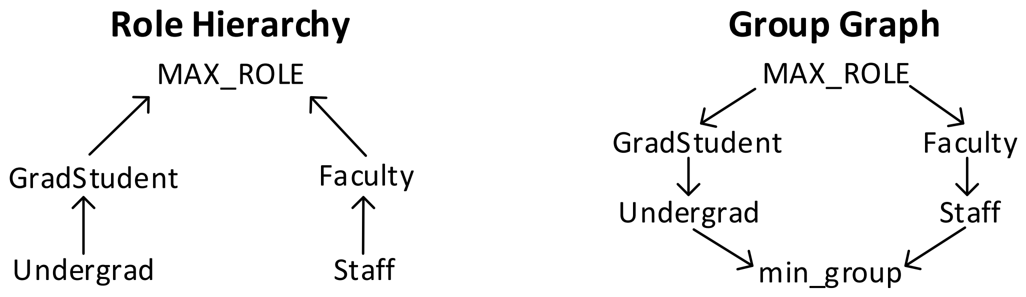


Figure 3.5: Example role hierarchy (left) and required group graph (right) to emulate it in HGABAC.

Each group is assigned a single attribute named "perms" that contains the set of permissions that group grants. Objects are tagged with an attribute for each access mode whose value contains the set of permissions that grant permission to perform that access mode on the object. For example, an object may have a "read" attribute with values p_1 and p_4 and a "write" attribute with values p_2 and p_3. The set of permissions are then simply: `PERMS = {("`*`user.perms IN object.read`*`", read), ("`*`user.perms IN object.write`*`", write)}` assuming the only access modes are `read` and `write`.

If the roles in the above example role hierarchy have the following (shown in Table 3.5) directly assigned permissions, then the groups in the user group graph will have the following direct and effective values for the attribute "perms":
While this enables HGABAC to emulate core and hierarchical RBAC (as defined in the NIST RBAC standard[Ferraiolo et al. 2001]), work towards emulating the separation of duty style constraints possible in NIST RBAC is left to future work.

Table 3.5: Resulting `direct(g)` and `effective(g)` attribute sets for each group (`g`) in the Group Graph given in Figure 3.5.

Role	Direct Permissions
Undergrad	P_1
Staff	P_2
GradStudent	P_3, P_4
Faculty	P_5, P_6
MAX_ROLE	$\emptyset$

`g`	`direct(g)`	`effective(g)`
min_group	$\emptyset$	$\emptyset$
$Undergrad$	P_1	P_1
$Staff$	P_2	P_2
$GradStudent$	P_3, P_4	P_1, P_3, P_4
$Faculty$	P_5, P_6	P_2, P_5, P_6
MAX_ROLE	$\emptyset$	P_1, P_2, P_3, P_4, P_5, P_6

3.6 Conclusions & Future Work

We have introduced a new model of ABAC, entitled HGABAC, that addresses a number of the open problems identified in Chapter 2 Section 2.4. Support for boolean rule based policies, hierarchical user and object groups, and a new administrative attribute simplify administration of ABAC systems and aiding in user comprehension of policies and attributes (an issue identified in Chapter 2 Section 2.4.9).

We show that adding hierarchical user and object groups enables greater flexibility when modelling real world situations in addition to providing a novel means of representing traditional access control policies (including hierarchical RBAC, DAC and MAC). Both the lack of hierarchical structures and the need for backwards compatibility are potential road blocks for ABAC acceptance identified in Chapter 2 Sections 2.4.2 and 2.4.3 that this work addresses.

Future work in terms of formalizing a model of ABAC should largely consist of extending HGABAC to support features required for real world use of ABAC systems. Some potential additions include support for separation of duty, delegation (addressed in Chapters 4 and 6), and access control for administrative functions. Expanding the policy language defined in Section 3.3.2 or alternatively exploring using XACML in its place could lead to greater flexibility in supported policies. To achieve the full potential of ABAC, further automation is needed in terms of attribute assignment and group membership. The addition of conditional user and object group membership could also have interesting applications and implications that are worthy of future research.

Further work is also needed towards providing supporting architectures and frameworks to aid in the implementation and use of HGABAC based systems in the real world. Chapter 5 explores this direction and provides an architecture for sharing attributes, describes the services needed for a distributed HGABAC based system and further extends the HGPL policy language to include attribute name spaces.

Finally, efforts towards a formal analysis of the safety of the HGABAC model and HGPL policy language are needed. In this direction, tools to aid policy administrators in verifying the correctness of policies and group hierarchies will be critical for further simplifying the administration and auditability of large ABAC systems. Such tools should provide a means of generating group hierarchies and HGPL policies while avoiding unintended side effects or interactions with existing policies and groups. Work towards formalisation of an HGABAC administrative model should also consider providing means of ensuring that multiple policy and group administrators can operate in the same system without inadvertently altering each others configurations.

Bibliography

D.E. Bell and L.J.L. Padula. *Secure Computer Systems: Mathematical Foundations and Model*. Mitre, 1974.

John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE, 2007.

Suroop Mohan Chandran and James BD Joshi. LoT-RBAC: A location and time-based RBAC model. In *Web Information Systems Engineering–WISE 2005*, pages 361–375. Springer, 2005.

Liang Chen and Jason Crampton. Risk-aware role-based access control. In *Proceedings of the 7th international conference on Security and Trust Management*, pages 140–156. Springer-Verlag, 2011.

Dave Crocker and Paul Overell. Augmented bnf for syntax specifications: Abnf. Technical report, RFC 2234, November, 1997.

Dorothy E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5): 236–243, May 1976. ISSN 0001-0782.

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.

Simon Godik and Tim Moses. OASIS extensible access control markup language (XACML). Technical report, OASIS Committee Secification cs-xacml-specification-1.0, 2002. URL https://www.oasis-open.org/committees/download.php/932/draft-xacml-schema-policy-13.pdf.

Xin Jin, Ram Krishnan, and Ravi Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. In *Data and Applications Security and Privacy XXVI*, pages 41–55. Springer, 2012.

Stephen Cole Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4): 150–155, 1938.

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

Butler W Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

Bo Lang, Ian Foster, Frank Siebenlist, Rachana Ananthakrishnan, and Tim Freeman. A flexible attribute based access control method for grid computing. *Journal of Grid Computing*, 7(2): 169–180, 2009.

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

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

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *The 7th International Symposium on Foundations and Practice of Security (FPS'2014)*, pages 187–204, November 2014.

Daniel Servos, Sabah Mohammed, Jinan Fiaidhi, and Tai-Hoon Kim. Extensions to ciphertext–policy attribute–based encryption to support distributed environments. *International Journal of Computer Applications in Technology*, 47(2):215–226, 2013.

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

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

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

Chapter 4

Strategies for Incorporating Delegation into Attribute-Based Access Control

Reprint Notice

A version of this chapter was published in Springer's Lecture Notes in Computer Science series and was part of the FPS'2016 proceedings as:

Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *The 9th International Symposium on Foundations and Practice of Security (FPS'2016)*, pages 320–328, October 2016
https://doi.org/10.1007/978-3-319-51966-1_21

Reprinted by permission from Springer Nature: Lecture Notes in Computer Science (LNCS, volume 8930) © 2016. Copyright permission and license can be found in Appendix A.3.

4.1 Introduction

Attribute-Based Access Control (ABAC) is a relatively new form of access control that bases access control decisions on the attributes of users, objects and the environment rather than the identity of users or the roles/clearances assigned to them. While there has been significant interest in the creation, enforcement and application of ABAC models[Jin et al. 2012; Servos and Osborn 2014], as of 2016 there are few works that address how delegation might be implemented or supported, leading to delegation to being identified as one of the major open problems (Chapter 2 Section 2.4.6) in current ABAC efforts.

Delegation enables a user to temporarily and dynamically alter the design of an access control system after policies have been created to account for everyday changes that policies are insufficient to address. In traditional models of access control delegation is relatively straightforward. A set of permissions or a role membership is delegated directly by a delegator to a delegatee under set conditions (e.g. an expiry date). In ABAC, this is complicated by both the introduction of attributes and ABAC's identity-less nature (i.e. access decisions are made on the basis of attributes and the user's identity may be unknown). Attributes may seem like an ideal access control element to build delegation around (as is done in ABE[Bethencourt et al.

2007; Servos et al. 2013] and Attribute Certificates[Turner et al. 2010]); however, as we will show, this naive approach comes with a number of unexpected challenges.

This chapter offers a preliminarily investigation into strategies for incorporating delegation into ABAC with the hope that they may lead to the development of full ABAC delegation models (such as the one presented in Chapter 6). Potential strategies are created by evaluating the combinations of delegators, delegatable access control elements and delegatees common in most ABAC models (Section 4.2.1). The trade-offs associated with each family of strategies are discussed and multiple examples are given that demonstrate how delegation might be performed (Section 4.2.2). Finally, we give conclusions and outline directions for future work (Section 4.4). It is our hope that this work will aid future research by identifying possible strategies for the creation of ABAC delegation models as well as the challenges and benefits associated with them.

4.2 Description of Potential Delegation Strategies for ABAC

4.2.1 Delegation Components

Delegation can be thought of as relating three access control components; a delegator, a delegatee and a delegatable access control element. A delegator temporarily grants a delegatee an access control element (e.g. a set of permissions or role membership) under set constraints. In RBAC delegation models, this is relatively straightforward: the delegator and delegatee are typically users and the access control element being delegated is either a set of permissions (via a temporary role)[Wang and Osborn 2011] or membership in an existing role[Barka et al. 2000]. ABAC, however, presents new possibilities for delegators, delegatees and delegatable elements that result in different trade-offs and limitations when combined. Each combination provides a conceivable strategy for delegation and offers particular advantages/disadvantages if used as the basis for an ABAC delegation model. This can be formulated as a Strategy Graph (as shown in Figure 4.1) in which each possible path from a delegator to a delegatee forms a potential strategy for incorporating delegation into ABAC.

Delegatable elements are the most important characteristic of delegation as they answer *what* is being delegated, while the delegators and delegatees answer *who* and *where* (i.e. *who* is doing the delegating and *where* the elements are being delegated to). The following are the most suitable delegatable elements that we have identified in current ABAC models[Jin et al. 2012; Servos and Osborn 2014]:

Attributes: Perhaps the most obvious element and one that has been explored to a limited extent (in ABE[Bethencourt et al. 2007; Servos et al. 2013] and Attribute Certificates[Turner et al. 2010]) are user attributes. In cases where attributes are delegatable, users are allowed to delegate their assigned attributes to a delegatee such that they are considered to be part of the delegatee's attribute set.

Permissions: Delegating permissions a delegator has obtained from a policy decision is another option. In such cases users are granted permissions as a result of their attribute set satisfying a policy and can delegate these permissions onto others while the policy remains satisfied.

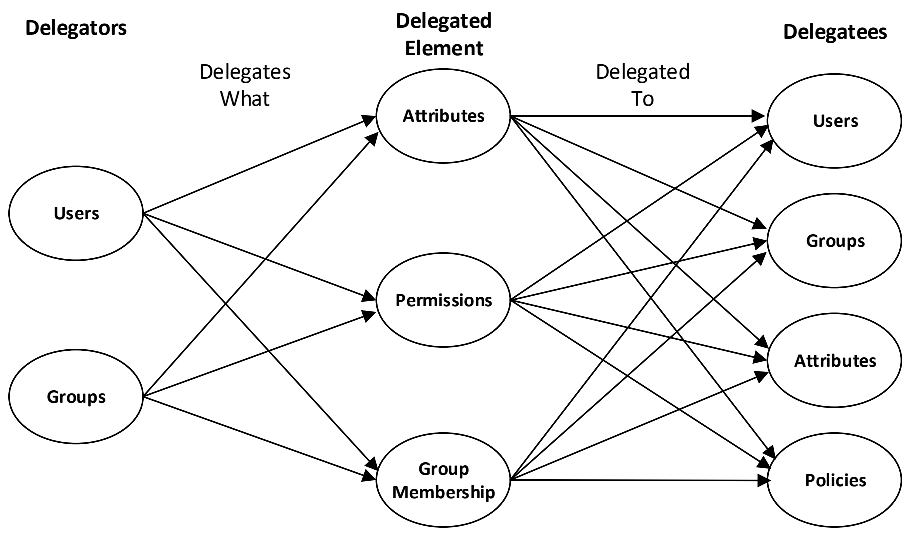


Figure 4.1: Strategy Graph

Group Membership: Recent ABAC models have incorporated the concept of user groups into the core ABAC model. In HGABAC[Servos and Osborn 2014] (Chapter 3), groups can be directly assigned user attributes that are inherited by users through their membership. Membership in these groups provides a possible delegatable element, similar to how role membership is delegatable in some RBAC delegation models[Barka et al. 2000].

While traditional models focus on delegation between users, additional possibilities exist for ABAC. In ABAC models with group support, user groups can be delegators in the sense that attributes or other delegatable elements assigned to groups may be temporarily delegated to a delegatee. In such a case, while the group is the source of the delegatable elements, the actual instigator of the delegation would be the members of the group or another actor in the system (e.g. a group leader). Similarly, the delegatee need not be limited to a user. Delegating to a group allows a delegator to assign their delegatable elements to multiple users in one operation. This is useful in scenarios where multiple users are briefly required to take on the duties of a single delegator (e.g. an absent store manager delegating his permissions to all department managers). In cases where group membership is being delegated, it can be considered that all members in the delegatee group are also temporarily made members of the delegated group.

Delegations can also be made to a policy or attribute. When an attribute is acting as a delegatee, all users that are directly (not through delegation) assigned the same attribute also become delegatees. For example if a permission, P, is delegated to the attribute *(ROLE, {manager})* (an attribute named *ROLE* with the value “manager”) all users that are assigned the attribute *ROLE* with a value of “manager” will be delegated the permission P. Using a policy as a delegatee works similarly. A delegator delegates some element to a policy they create and all users satisfying this policy are delegated the element. For example, if membership in a group, G, is delegated to the policy `ROLE = manager AND YEARS_EMPLOYED ≥ 3`, users that have attributes stating that they are managers and employed for at least 3 years will be delegated membership in group G. While delegating to an attribute or policy may seem complex, it is a necessity to support delegation in a system where the identity of a user may remain unknown and access decisions are made purely on the user’s attributes.

Table 4.1: Delegation Strategies

Strategy Name	**Delegator**	**Delegatable Element**	**Delegatee**
Attribute Delegation			
User-to-User Attribute Delegation	U	AS	U
User-to-Group Attribute Delegation	U	AS	G
Group-to-Group Attribute Delegation	G	AS	G
Group-to-User Attribute Delegation	G	AS	U
User-to-Attribute Attribute Delegation	U	AS	A
Group-to-Attribute Attribute Delegation	G	AS	A
User-to-Policy Attribute Delegation	U	AS	P
Group-to-Policy Attribute Delegation	G	AS	P
Group Membership Delegation			
User-to-User Membership Delegation	U	GM	U
Group-to-User Membership Delegation	G	GM	U
Group-to-Group Membership Delegation	G	GM	G
User-to-Group Membership Delegation	U	GM	G
User-to-Attribute Membership Delegation	U	GM	A
Group-to-Attribute Membership Delegation	G	GM	A
User-to-Policy Membership Delegation	U	GM	P
Group-to-Policy Membership Delegation	G	GM	P
Permission Delegation			
User-to-User Permission Delegation	U	PS	U
User-to-Group Permission Delegation	U	PS	G
Group-to-User Permission Delegation	G	PS	U
Group-to-Group Permission Delegation	G	PS	G
User-to-Attribute Permission Delegation	U	PS	A
Group-to-Attribute Permission Delegation	U	PS	A
User-to-Policy Permission Delegation	U	PS	P
Group-to-Policy Permission Delegation	G	PS	P

Legend	
U = User	**PS** = Policy Set
G = Group	**AS** = Attribute Set
P = Policy	**GM** = Group Membership
A = Attribute	

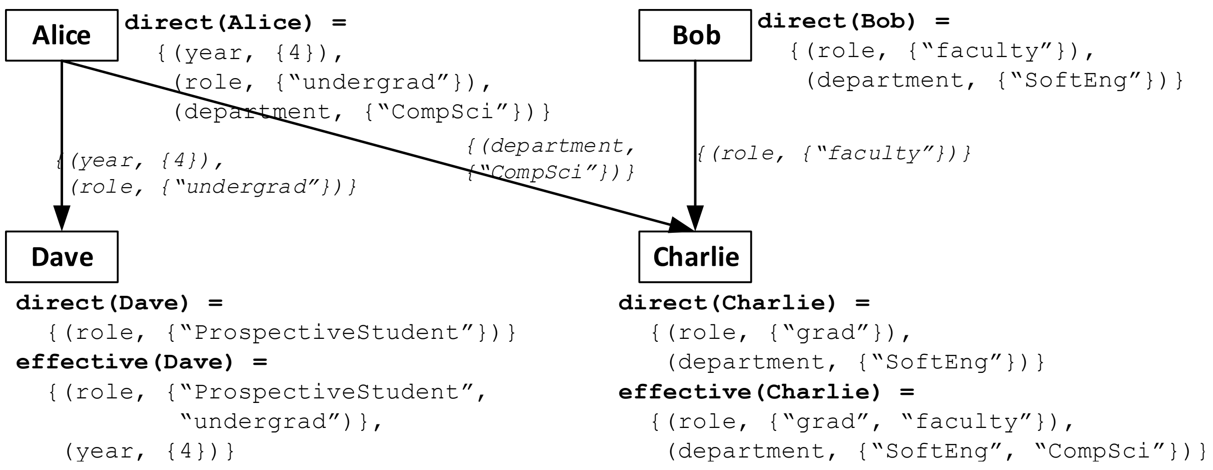


Figure 4.2: Example of User-to-User Attribute Delegation. Arrows denote direction of delegation (arrow points to delegatee), boxes represent users of the system.

4.2.2 Delegation Strategies

Figure 4.1 shows how each delegation component described in Section 4.2.1 may be combined to create a delegation strategy for ABAC. Each path from a possible delegator component to a possible delegatee component in the graph represents a feasible strategy. For example the path *(Users, Permissions, Users)* represents a strategy in which users can delegate their permissions to other users, whereas *(Groups, Attributes, Policies)* represents a strategy in which groups can delegate their attributes to other users if they satisfy a given policy. Table 4.1 categorizes each strategy into families based on the element being delegated. Strategies in the same family tend to share common characteristics and challenges for systems adopting them. In this section, we discuss the advantages and limitations of each family. It is assumed that only one strategy is used at a time. A delegation model for ABAC will need to utilize one or more of these strategies in addition to traditional delegation concepts (constraints, revocation, etc.).

4.2.2.1 Attribute Delegation

In Attribute Delegation strategies, delegatees are delegated a subset of the delegator's attributes. Delegated attributes are merged with the delegatee's directly assigned attributes (i.e. assigned through any means but delegation) and the combined attribute set is treated as the delegatee's set during policy evaluation. An example of User-to-User Attribute Delegation is shown in Figure 4.2 where *direct*(*user*) is the user's directly assigned attributes and *effective*(*user*) is the user's effective attributes (i.e. the merged attribute set used for policy evaluations). In Figure 4.2, Alice wants to delegate a subset of her attributes to a prospective student (Dave) so he can satisfy the policy `role = "undergrad" AND year` $\geq$ `2` to view some resource. As Dave only has the value "ProspectiveStudent" for his *role* attribute and no *year* attribute, Alice must delegate both her *role* and *year* attributes for Dave to satisfy the policy. The subset Alice delegates is {*(year,* {*4*}*), (role,* {*"undergrad"*}*)*} which makes Dave's effective attribute set {*(role,* {*"ProspectiveStudent", "undergrad"*}*)*}*, (year,* {*4*}*)*}.

Multiple simultaneous delegations to a single user are also possible. In Figure 4.2, Alice wishes to delegate to Charlie so he can satisfy the policy `role IN { "undergrad", "grad"}`

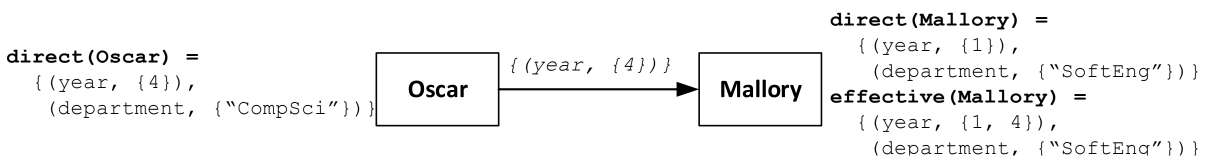


Figure 4.3: Example of a possible attack on User-to-User Attribute Delegation.

`AND department = "CompSci"`, and access a resource limited to CompSci students. At the same time, Bob wishes to delegate to Charlie so he can satisfy the policy `role = "faculty" AND department = "SoftEng"` and access a resource limited to SoftEng faculty. Alice delegates {*(department,* {*"CompSci"*}*)*} and Bob {*(role,* {*"faculty"*}*)*}. Making Charlie's effective attributes {*(role,* {*"grad", "faculty"*}*), (department,* {*"SoftEng", "CompSci"*}*)*}.

While this style of delegation is easy to implement (a subject's effective attribute set is simply used in place of their direct set), it can lead to serious problems if not carefully constrained. The first issue is the creation of conflicting policy evaluations. In Figure 4.2 Alice's delegation results in Dave's effective attribute set containing two values for the *role* attribute, "ProspectiveStudent" and "undergrad". If a policy were to exist such as `role ≠ "ProspectiveStudent"` two different results would be possible depending on the value of *role* used when evaluating the policy. A potential solution is to use a policy language that specifies clear resolutions to conflicts (e.g. prioritize attributes assigned via delegation over those directly assigned or always grant access when any combination of attributes satisfies the policy). However, the issue is further complicated when multiple delegations to the same delegatee are considered simultaneously. In such cases, conflicts can arise from purely delegated attributes, making conflict handling more difficult (e.g. can not simply prioritize delegated attributes).

A second issue is the potential for users to collude to satisfy a policy that they would individually be unable to. In Figure 4.3 Oscar and Mallory are trying to satisfy the policy `year > 2 AND department = "SoftEng"`. Individually, neither can satisfy the policy as Oscar lacks a *department* attribute with a "SoftEng" value and Mallory lacks a *year* attribute with a value greater than 2. However, if Oscar delegates {*"year",* {*4*}} to Mallory it creates the effective attribute set {*(year,* {*1, 4*}*), (department,* {*"SoftEng"*}*)*} and Mallory can satisfy the policy if *year* is evaluated as 4. While one solution is to heavily constrain what attributes can be delegated or to use a constraint specification language[Bijon et al. 2013] to enforce SoD style constraints, the simplest fix is to isolate delegated attribute sets from each other and the delegatee's directly assigned set. Thus, a user must choose what set of attributes to activate at the start of a session (similar to role activation in RBAC[Ferraiolo et al. 2001]). Isolation of attribute sets would also provide a solution to conflicting policy evaluations and aid in user comprehension. For example, Alice would know that if she delegates all of her attributes to Dave, at most Dave would have access to the same permissions as he did before in addition to the permissions Alice has access to. Users would still be able to bypass negative polices like *"year ≠ 4 AND year ≠ 1"* if not having a *year* attribute is considered to satisfy the policy by delegating a subset of their attributes that omits the *year* attribute.

A third issue resulting from merging attribute sets is losing the descriptiveness of the delegatee's attributes. In Figure 4.2, after delegation, Dave's effective attribute set is no longer descriptive of Dave. Dave obtains a *year* attribute with a value of 4 while not being a student.

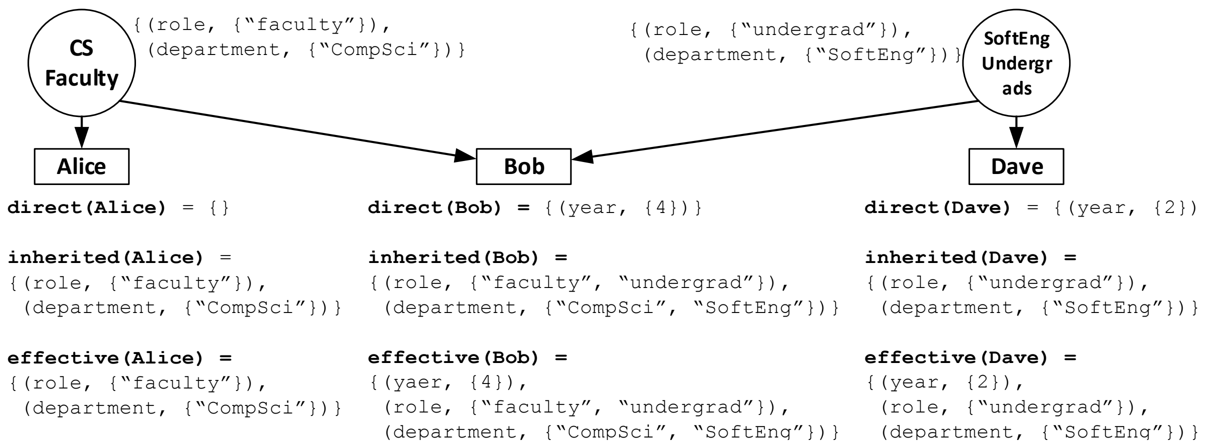

CS Faculty
{(role, {"faculty"}), (department, {"CompSci"})}
SoftEng Undergrads
{(role, {"undergrad"}), (department, {"SoftEng"})}
Alice
Bob
Dave
direct(Alice) = {}
inherited(Alice) = {(role, {"faculty"}), (department, {"CompSci"})}
effective(Alice) = {(role, {"faculty"}), (department, {"CompSci"})}
direct(Bob) = {(year, {4})}
inherited(Bob) = {(role, {"faculty", "undergrad"}), (department, {"CompSci", "SoftEng"})}
effective(Bob) = {(yaer, {4}), (role, {"faculty", "undergrad"}), (department, {"CompSci", "SoftEng"})}
direct(Dave) = {(year, {2})
inherited(Dave) = {(role, {"undergrad"}), (department, {"SoftEng"})}
effective(Dave) = {(year, {2}), (role, {"undergrad"}), (department, {"SoftEng"})}

Figure 4.4: Example of attribute user groups from HGABAC[Servos and Osborn 2014]. User groups are shown as circles and users as rectangles. Arrows denote a user being a member of a group.

While this makes delegation possible and allows Dave to satisfy the policy, it complicates policy creation (need to account for unexpected attribute combinations) and restricts the use of attributes to the purpose of access control (e.g. a system could not trust that an e-mail sent to an address in a user's effective attribute set was actually theirs).

The last issue is comprehension of what is being delegated and what needs to be delegated to achieve a desired result. A delegator must be familiar with the policies of the system and their own attributes. In Figure 4.2, if Alice wanted to delegate a permission she was granted from satisfying the policy `role = "undergrad" AND year` $\geq$ `2` she would have to understand the policy, what attribute set she has been assigned and what attribute subset to delegate. This is further complicated if delegated attribute sets are not isolated, as Alice would also have to be aware of possible conflicts and unexpected attribute combinations.

4.2.2.2 Group Membership Delegation

Group Membership Delegation requires an ABAC model which supports user groups in which members of a group inherit attributes assigned to that group. Figure 4.4 shows an example of how user groups work in HGABAC[Servos and Osborn 2014]. In this case Alice and Bob are members of the CS Faculty group and inherit the attributes *role* and *department* with values "faculty" and "CompSci" respectively. Additionally, Bob is a member of the SoftEng Undergrad group and inherits the values "undergrad" and "SoftEng" for the attributes *role* and *department*. These inherited attributes are merged with the user's directly assigned attributes to form the user's effective attribute set (similar to how attributes are merged in Attribute Delegation). In Group Membership Delegation, membership in groups are delegated as opposed to the delegator's attributes. In Figure 4.4, if Alice wanted to delegate a permission she was granted from belonging to the CS Faculty group (e.g. from satisfying the policy `role = "faculty" AND department = "CompSci"`) to Dave she would delegate her membership in the CS Faculty group such that Dave's inherited set of attributes would be {*(role,* {*"undergrad", "faculty"*}*), (department,* {*"SoftEng", "CompSci"*}*)*} leading to the effective attribute set {*(year,* {2}*), (role,* {*"faculty", "undergrad"*}*), (department,* {*"CompSci", "SoftEng"*}*)*} when merged with his attributes.

This method of delegation has several advantages over Attribute Delegation. User comprehension is improved as users are not required to pick individual attributes to delegate and instead only need to consider what group memberships are needed. Placing constraints on delegation becomes easier as delegators are forced to delegate whole attribute sets belonging to groups at a time (constraints can be placed on what group memberships can be delegated and by whom, rather than individual attributes). Finally, the effective attribute set of delegatees is more likely to remain descriptive of the delegatee as personal attributes (like year, age, etc.) are more likely to be directly assigned than assigned to groups.

Despite these advantages, Group Membership strategies share a number of issues in common with Attribute Delegation. Conflicting policy evaluations and user collusion is still possible, although more restrained. For collusion to be possible, groups have to be assigned the required attribute value pairs. For example, if the policy was `role = "faculty" AND department = "SoftEng"`, Alice and Dave could still collude to satisfy the policy (by Alice delegating her membership in the CS Faculty group to Dave); however, it would not be possible for Alice and Dave to collude to satisfy the policy `year > 1 AND department = "CompSci"` as *year* is a directly assigned attribute. Isolating attribute sets obtained through membership delegation and attribute sets obtained through normal assignment would minimize the issue and avoid unforeseen permissions being granted (e.g. if Alice delegates her membership a group to Dave, she knows that Dave would not satisfy any policy that she her self could not satisfy from her membership).

Group Membership Delegation also introduces a new issue. Attributes that are directly assigned to a delegator, like the *year* attribute in Figure 4.4, are undelegatable. Assuming this attribute is only directly assigned to users and never to groups, it would be impossible to delegate membership to satisfy a policy such as `year ≥ 2`. A system utilizing Group Membership Delegation would either have to carefully design its groups such that all desired delegation use cases can be accomplished through delegating group memberships or implement a second delegation strategy in addition to Group Membership Delegation.

4.2.2.3 Permission Delegation

Rather than delegating attributes (directly or indirectly) Permission Delegation strategies are based on delegating permissions. Delegators are able to delegate permissions they obtain by satisfying policies onto delegatees so long as the granting policy remains satisfied (e.g. if the delegator's attributes or an environmental attribute changes such that the policy granting the permission is no longer satisfied, the delegated permission is revoked). In strategies where a group is the delegator, the permissions the group can delegate is equal to the set of permissions a user would be granted if they had the same attributes as the group. For example, if the users and groups from Figure 4.4 and the policy `role = "faculty" AND department = "CompSci"` existed that granted the permission, p_1, both Alice and Bob as well as the group CS Faculty could delegate p_1. If the policy `year ≥ 2 AND TIME > 9:00AM AND TIME < 5:00PM` granted the permission p_2, Bob and Dave could delegate p_2 but the delegation would only be valid between 9:00AM and 5:00PM.

Permission Delegation strategies poses greater challenges in terms of implementation but resolve the issues faced by the other families. As delegated permissions are only valid while the policy granting them remains satisfied, a system would be required to either periodically check that the delegator still satisfies the policy or recheck the policy each time the delegatee uses the permission. Depending on the size of the system and the complexity of the policies, this could add significant overhead. The benefit is that no change is made to the delegatee's attribute set, limiting conflicting policy evaluations and preventing user collusion. User comprehension is also improved as users are delegated permissions directly rather than attributes that only indirectly grant permissions.

4.2.2.4 XACML & Policy Rights Delegation

The XACML Administration and Delegation Profile[Rissanen et al. 2009] enables the support of attribute-based administration and delegation policies in XACML. This is accomplished by adding permissions that grant the right to issue new policies under certain constraints and delegatees are able to further delegate these rights on to other delegatees so long as the same constraints are maintained. For example, a system using this style of delegation might have a trusted policy (e.g. one created by a system administrator) that grants Alice the right to create a policy that grants access to use a printer so long as the subject using the printer has a *role* attribute with the value “employee”. This policy would not grant Alice (or any one else) the right to use the printer (even if she had the required attribute value pair), but only to create a policy under these constraints. If Alice wished to allow Bob access to the printer she could create an access policy that stated that a user with the name Bob could use the printer so long as he had a *role* attribute with the value “employee”. She could also create a policy that allowed anyone with a *year* attribute with a value greater then 2 and a *role* attribute with the value “employee” to use the printer, to allow any user that meets these conditions to use the printer. However, Alice could not create a policy that lacks the requirement that the user be an employee as this is a constraint in the original delegation policy.

While this style of delegation is well suited for administration purposes, it presents some significant user comprehension barriers. Users wishing to temporarily delegate a permission to another user must have adequate understanding of policy creation, the policy under which they are permitted to delegate, and the attributes that exist in the system. Allowing users to create policies also poses potential risks if not properly constrained. As mentioned in [Rissanen et al. 2009], it may be possible for a user to create policies such that a significant amount of computation will be required during policy evaluation to the point of degrading the performance of the system (effectively executing a denial of service attack).

4.2.3 Revocation

Revocation is an integral component of delegation models that provides a means to undo or revoke the dynamic delegations that have been made during the execution of the system. In traditional access control models, revocation is commonly described by two properties cascading/non-cascading and grant dependent/grant independent. Cascading revocation applies to delegation models that support multiple levels of delegation. For example, Alice might delegate permissions p_1 and p_2 to Bob, who further delegates p_2 onto Dave who in turn delegates p_2 to Charlie. In cascading revocation, if Alice revoked her delegation of p_1 and p_2 to Bob, the whole chain of delegation is broken and Dave and Charlie would lose access to p_2. In non-cascading revocation only Bob would lose access to the delegated permissions, while

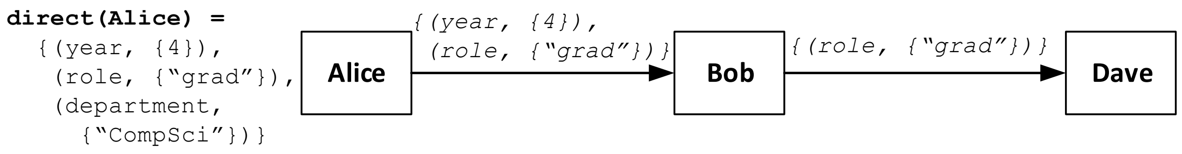


Figure 4.5: Example User-to-User Attribute Delegation chain.

Charlie and Dave would maintain access to p_2. Grant dependence describes who may revoke a delegated permission. In grant dependent revocation, only the delegator who invoked the original delegation can revoke it. In grant independent revocation any delegator that is able to make the same delegation may revoke it. For example, if Alice delegated p_1 and p_2 to Bob, and Dave is also able to delegate p_1 and p_2 to Bob (but has not in this case), both Alice and Dave can revoke Alice's delegation of p_1 and p_2 to Bob. In addition to user enacted revocation, revocation can also happen as the result of some constraint being violated (e.g. a time limit placed on the delegation expires or a user is no longer a member of a required group or role).

In attribute-based systems, revocation is further complicated as several ABAC frameworks and architectures[Yuan and Tong 2005; Hu et al. 2013] isolate the attribute granting and policy evaluation parts of the system. For example in the ABAC security architecture described by Yuan and Tong[Yuan and Tong 2005], attribute authorities issue attributes to users in the form of SAML Attribute Assertions[Ragouzis et al. 2008] that are passed on to services the user desires to make requests upon (the services then in turn use the policy decision service to determine if the request is granted). These attribute assertions are cryptographically signed documents that detail a user's attribute set and the conditions under which they are valid. This is problematic for revocation, however, as these assertions are immutable after they have been issued and there may be no way to directly communicate changes that would initiate revocation between the attribute authorities and the policy decision service. Figure 4.5 shows a case where this can be an issue. In this example of User-to-User Attribute Delegation, Alice delegates the attribute subset {*(year,* {*4*}*), (role,* {*"grad"*}*)*} to Bob and Bob further delegates a subset of these attributes, {*(role,* {*"grad"*}*)*}, onto Dave. If Dave requests his attributes from an attribute authority he will be issued an assertion for the attribute set {*(role,* {*"grad"*}*)*} that is valid for some time period. However, if Alice or Bob wished to revoke their delegation before this time period had elapsed they would have no way of easily doing so.

While solutions such as revocation lists and sending notices between the attribute authorities and the policy decision service are possible, they require some level of direct communication which undermines many of the benefits of this kind of ABAC security architecture. An alternative such as having the policy decision service directly contact the attribute authority for the user's attribute set rather than using a SAML assertion indirectly passed through the user would work but would eliminate benefits such as single sign-on, interoperability between isolated attribute authorities and policy decision services run by different organizations, ability to use an off-line attribute authority and in some cases scalability would be limited by requiring direct communication. For these reasons, the underlying ABAC security architecture used will have a great impact on what methods of revocation are possible or practical for each delegation strategy.

Table 4.2: Evaluation of Delegation Strategies

Strategy	Requires Features	User Comprehension	Attributes Remain Descriptive	Conflicting Policy Evaluations	Persistent Evaluation Required
Attribute Delegation					
User-to-User	Core ABAC	Low	No	Yes	No
User-to-Group	Core ABAC	Low	No	Yes	No
Group-to-Group	Core ABAC, User Groups	Low	Depends on Group	Yes	No
Group-to-User	Core ABAC, User Groups	Low	Depends on Group	Yes	No
User-to-Attribute	Core ABAC	Low	No	Yes	No
Group-to-Attribute	Core ABAC, User Groups	Low	Depends on Group	Yes	No
User-to-Policy	Core ABAC	Very Low	No	Yes	Yes
Group-to-Policy	Core ABAC, User Groups	Very Low	Depends on Group	Yes	Yes
Group Membership Delegation					
User-to-User	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
Group-to-User	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
Group-to-Group	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
User-to-Group	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
User-to-Attribute	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
Group-to-Attribute	Core ABAC, User Groups	Medium	Depends on Group	Yes	No
User-to-Policy	Core ABAC, User Groups	Low to Medium	Depends on Group	Yes	Yes
Group-to-Policy	Core ABAC, User Groups	Low to Medium	Depends on Group	Yes	Yes
Permission Delegation					
User-to-User	Core ABAC	High	Yes	No	Yes
User-to-Group	Core ABAC	High	Yes	No	Yes
Group-to-User	Core ABAC, User Groups	High	Yes	No	Yes
Group-to-Group	Core ABAC, User Groups	High	Yes	No	Yes
User-to-Attribute	Core ABAC	High	Yes	No	Yes
Group-to-Attribute	Core ABAC, User Groups	High	Yes	No	Yes
User-to-Policy	Core ABAC	Medium to High	Yes	No	Yes
Group-to-Policy	Core ABAC, User Groups	Medium to High	Yes	No	Yes
XACML/Policy Rights Delegation					
XACML Profile	Core ABAC, XACML	Low to Medium	Yes	No	Yes

Strategy	Possible Revokers	Revocation Complexity	Implementation Complexity
Attribute Delegation			
User-to-User	Delegator	Low	Very Low
User-to-Group	Delegator	Low	Low
Group-to-Group	Delegator, Group Member	Low	Low
Group-to-User	Delegator, Group Member	Low	Low
User-to-Attribute	Delegator	Medium	Medium
Group-to-Attribute	Delegator, Group Member	Medium	Medium
User-to-Policy	Delegator, Policy	Medium to High	Medium
Group-to-Policy	Delegator, Group Member, Policy	Medium to High	Medium
Group Membership Delegation			
User-to-User	Delegator	Low	Very Low
Group-to-User	Delegator, Group Member	Low	Low
Group-to-Group	Delegator, Group Member	Low	Low
User-to-Group	Delegator	Low	Low
User-to-Attribute	Delegator	Medium	Medium
Group-to-Attribute	Delegator, Group Member	Medium	Medium
User-to-Policy	Delegator, Policy	Medium to High	Medium
Group-to-Policy	Delegator, Group Member, Policy	Medium to High	Medium
Permission Delegation			
User-to-User	Delegator, Policy	Medium to High	Medium
User-to-Group	Delegator, Policy	Medium to High	Medium to High
Group-to-User	Delegator, Group Member, Policy	Medium to High	Medium to High
Group-to-Group	Delegator, Group Member, Policy	Medium to High	Medium to High
User-to-Attribute	Delegator, Policy	High	High
Group-to-Attribute	Delegator, Group Member, Policy	High	High
User-to-Policy	Delegator, Policy	Very High	High
Group-to-Policy	Delegator, Group Member, Policy	Very High	High
XACML/Policy Rights Delegation			
XACML Profile	Unclear?	Unclear? Likely High	High

4.3 Qualitative Evaluation of Delegation Strategies

To further explore the advantages and limitations of each delegation strategy and summarize the issues discussed in Section 4.2, we have devised a number of qualitative attributes to evaluate the trade-offs associated with each strategy:

Required Features: The features (namely user group support) required by the underlying ABAC model to support this type of delegation.

User Comprehension: How accessible the concept of delegation is to users of the system. Will the users be able to easily understand the implications of their delegation and what rights are gained from it.

Attributes Remain Descriptive of Subject: How descriptive a subject's attributes remain of the subject after delegation occurs.

Potential for Conflicting Policy Evaluations: If the delegation could cause a conflict in the evaluation of an access control policy present in the system.

Persistent Evaluation of Policies Required: If one or more policies need to be continuously or periodically evaluated to maintain the delegation.

Revocation Complexity: The complexity of revoking a delegated permission, attribute, etc. before it expires.

Possible Revokers: The set of actors in the system that are able to revoke a delegated permission, attribute, etc. before the expiry date (assuming a grant dependent model of revocation).

Implementation Complexity: The difficulty or complexity of integrating this delegation strategy into an existing ABAC system or model.

Table 4.2 compares each strategy based on the above qualitative attributes. The ideal strategy largely depends on the needs and requirements of the implementing system; however, a few generalizations can be made. Permission Delegation strategies are suitable for systems requiring high user comprehension or wishing to remove the possibility of conflicting policy evaluations and removing the possibility of user collusion. Attribute Delegation strategies are appropriate when it is not possible to continually evaluate policies as is required in Permission Delegation or low implementation complexity is desired. Group Membership Delegation strategies provide higher user comprehension with similar results to Attribute Delegation but require user group support (lacking in many ABAC models). Finally, Policy Rights Delegation may provide an ideal solution to policy administration but its use for dynamic delegation may cause issues with user comprehension and open denial of service type vulnerabilities if policy creation is not strictly constrained.

From this summary, it can also been seen that X-to-Policy style strategies introduce higher revocation complexity and lower user comprehension. However, these strategies also provide greater flexibility and allow for delegation to users whose identity may not be known at the time of delegation or policy creation. X-to-Group strategies provide a means for delegators to delegate to groups of users in one operation but are dependent on user group support in the underlying ABAC model. X-to-Attribute strategies provide a middle ground between X-to-User and X-to-Policy strategies with less flexibility than the X-to-Policy strategies but increased user comprehension and retaining the ability to delegate to users that are not known at the time of delegation.

4.4 Conclusions & Future Work

4.4.1 Delegation Strategies

The delegation strategies described in this chapter seek to start addressing the open problem of delegation (Chapter 2 Section 2.4.6) by providing a number of directions for future ABAC based delegation models to explore. Each strategy discussed is a potential basis for creating a delegation model and come with their own advantages and disadvantages.

The ideal delegation strategy depends on the needs of the implementing system; however, a few generalizations can be made. Permission Delegation is suitable for systems requiring high user comprehension and removes the possibility of conflicting policy evaluations and user collusion. Attribute Delegation is ideal when continual policy evaluation would be difficult or low implementation complexity is desired. Group Membership Delegation provides high user comprehension with similar results to Attribute Delegation but requires group support.

Delegating to a user (X-to-User strategies) provides the closest parallel to delegation in traditional models, however, delegating to groups (X-to-Group), attributes (X-to-Attribute) or policies (X-to-Policy) can provide greater flexibility and allow for delegation to users whose identity is unknown during policy creation. X-to-Group allows for delegation to groups of users in one operation but requires group support. X-to-Policy introduces higher revocation complexity and lower user comprehension but has the greatest flexibility. X-to-Attribute provides a middle ground between the two with less flexibility than X-to-Policy but increases user comprehension while retaining the identity-less nature of ABAC.

4.4.2 Future Work

A number of directions are possible for future work. Using multiple strategies simultaneously could provide new possibilities for delegation. Such combinations could help overcome the limitations of individual strategies but further work is needed to evaluate any complexities or conflicts introduced. Existing policy conflict resolution techniques could help mitigate the issues faced by Attribute and Group Membership Delegation, as well as allow for hybrid strategies with minimal limitations. Additional work is required to determine if current techniques are applicable.

Formalizing the strategies described in this work will allow for in-depth analysis and aid integration into existing ABAC models. Extending an existing model with each strategy would allow for a more quantitative evaluation and provide a reference model for future work. HGABAC is an ideal candidate for such extensions by virtue of its support for user groups. This direction is explored in Chapter 6 by formalizing the User-to-User Attribute Delegation strategy as a new delegation model for HGABAC and a set of extensions to the HGAA architecture (described in Chapter 5), including an updated attribute certificate format that supports “off-line” delegation between users without a third party.

Bibliography

Ezedin Barka, Ravi Sandhu, et al. A role-based delegation model and some extensions. In *NISSC'00*, pages 396–404, 2000.

John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *SP'07*, pages 321–334, 2007.

Khalid Zaman Bijon, Ram Krishman, and Ravi Sandhu. Constraints specification in attribute based access control. *Science*, 2(3):131–144, 2013.

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

Vincent C Hu, David Ferraiolo, Rick Kuhn, et al. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication*, 800, 2013.

Xin Jin, Ram Krishnan, and Ravi S Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

Nick Ragouzis, John Hughes, Rob Philpott, Eve Maler, Paul Madsen, and Tom Scavo. Security assertion markup language (SAML) v2.0 technical overview. *OASIS Comittee Draft*, 2, 2008.

E Rissanen, H Lockhart, and T Moses. XACML v3. 0 administration and delegation profile version 1.0. *OASIS Standard*, 1, 2009.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *FPS'14*, pages 187–204, 2014.

Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *The 9th International Symposium on Foundations and Practice of Security (FPS'2016)*, pages 320–328, October 2016.

Daniel Servos, Sabah Mohammed, Jinan Fiaidhi, and Tai hoon Kim. Extensions to ciphertext-policy attribute-based encryption to support distributed environments. *IJCAT*, 47(2-3):215–226, 2013.

Sean Turner, Russ Housley, et al. An internet attribute certificate profile for authorization. RFC 5755, January 2010. URL https://tools.ietf.org/html/rfc5755.

He Wang and Sylvia L Osborn. Static and dynamic delegation in the role graph model. *IEEE TKDE*, 23(10):1569–1582, 2011.

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

Chapter 5

HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control

Reprint Notice

A version of this chapter was published in the proceedings of the Third ACM Workshop on Attribute-Based Access Control as:

Daniel Servos and Sylvia L Osborn. HGAA: An architecture to support hierarchical group and attribute-based access control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18)*, pages 1–12, March 2018

https://doi.org/10.1145/3180457.3180459

5.1 Introduction

Attribute-Based Access Control (ABAC) is an emerging form of access control that bases access control decisions on the attributes of users, objects and the environment rather than the identity of users or the roles/clearances assigned to them. While the beginnings of ABAC in academic literature can be seen as early as 15 years ago[Wang et al. 2004], ABAC has only recently gained significant attention in the past half decade[Servos and Osborn 2017]. This newfound interest has resulted in the creation of numerous ABAC models[Servos and Osborn 2014; Jin et al. 2012; Ferraiolo et al. 2011; Rubio-Medrano et al. 2013], however, none to date have gained acceptance as a unified standard or provided a complete view of how they might be implemented in practice. Real-world implementation details and results are still needed and existing models largely lack architectural specifications required for actual use and empirical study.

In our pervious work[Servos and Osborn 2014] (Chapter 3), we introduced Hierarchical Group and Attribute-Based Access Control (HGABAC), a model of Attribute-Based Access Control that incorporates hierarchical user and object groups to ease administration and increase policy flexibility. Since publication, extensions to and expansion of the original HGABAC model have been explored by ourselves and others, including introduction of an adminstative model ($GURA_G$)[Gupta and Sandhu 2016], the creation of an authorization architecture for a restricted HGABAC model (rHGABAC)[Bhatt et al. 2017] and preliminary work towards incorporating delegation concepts[Servos and Osborn 2016] (Chapter 4). However, none of these works provide a complete architecture to facilitate real-world implementation and use of HGABAC in a distributed environment or address the open issue of attribute storage and sharing (an open problem discussed in Chapter 2 Section 2.4.7).

Questions like “who assigns the attributes?”, “how are attributes shared with each party?”, “how does the user provide proof of attribute ownership?”, “where and how are policies evaluated?”, “how will the model scale in real-world use?”, etc. remain unanswered. This chapter attempts to answer these questions and more in regards to HGABAC through the creation of an attribute-based architecture, entitled Hierarchical Group Attribute Architecture (HGAA). HGAA enables the use of HGABAC in distributed environments by formalizing and providing the following key components:

- **Attribute Authority**: A service for managing, storing and providing user attributes in the form of attribute certificates that are used to authenticate with a user service provider.
- **Attribute Certificate**: A cryptographically secured certificate that details attributes a user has activated for a given session as well as revocation and delegation rules under which the certificate was issued.
- **HGABAC Namespace**: A URI-based namespace for uniquely identifying HGABAC elements (attributes, users, objects, etc.) across disparate security domains and authorities.
- **Policy Authority**: A service which manages and evaluates HGABAC policies on behalf of a user service provider.
- **User Service Provider**: A provider of services to end users that have access restricted on the basis of one or more HGABAC policies.

These service components are implemented as JSON-based web services and performance results are given to demonstrate the scalability of the architecture in terms of number of attributes and policy complexity. A detailed attribute certificate format is specified that includes support for future delegation and revocation extensions as discussed in [Servos and Osborn 2016] (Chapter 4) and utilized in Chapter 6 to create a HGABAC and HGAA based delegation model.

The remainder of this chapter is laid out as follows; Section 5.2 briefly introduces HGABAC and gives background information, Section 5.3 discusses related work and its applicability to HGAA, Section 5.4 introduces the overall architecture and details each component, Section 5.5 details our HGAA implementation and gives preliminary performance results and finally Section 5.6 presents our concluding remarks and considers directions for future work.

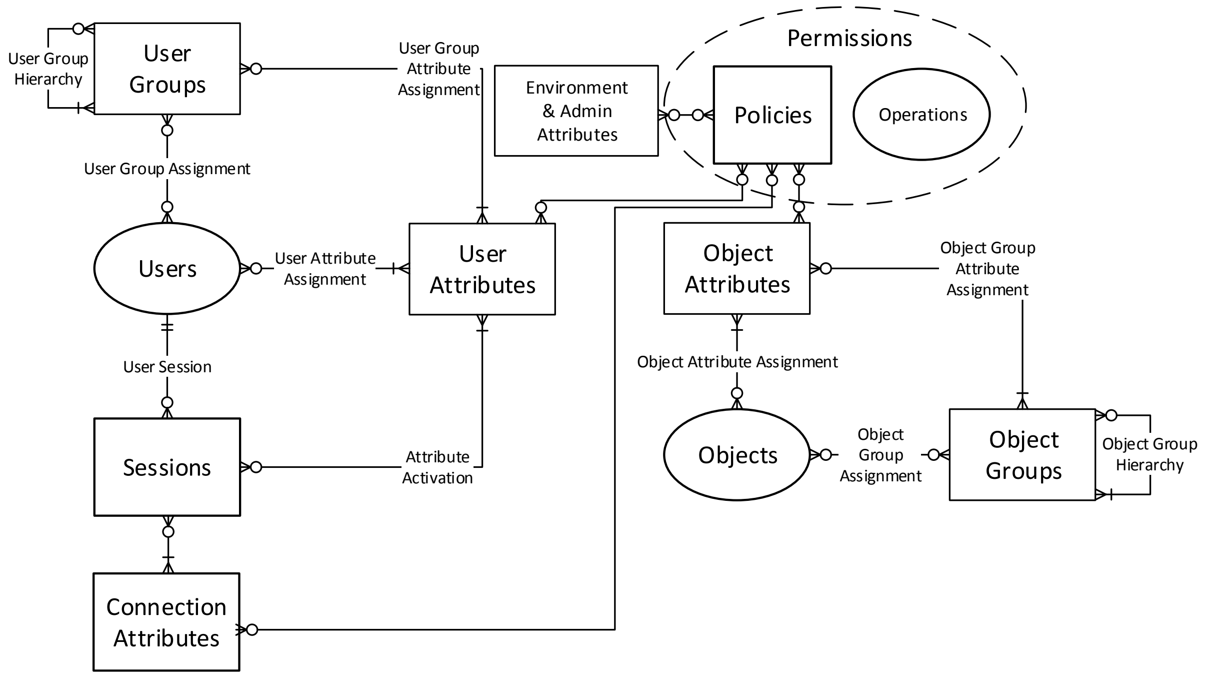


Figure 5.1: HGABAC components and relations using Crow's Foot Notation to denote cardinality of relationships. Primitive components are shown in ovals.

5.2 HGABAC Background

HGABAC[Servos and Osborn 2014] (Chapter 3) provided a formal model of ABAC that introduced group based hierarchical representations of object and user attributes that was not available in other models at the time. In HGABAC, attributes are assigned both directly to access control entities and indirectly assigned through user and object attribute groups (these relations are shown in Figure 5.1). Attribute groups help simplify administration of ABAC systems by allowing administrators to create user or object groups whose membership indirectly assigns sets of attribute/value pairs to its members. These groups are hierarchical and inherit attribute/value pairs from their parent groups allowing for more flexible policy representation when combined with the HGABAC policy language.

The group hierarchy is represented as a directed acyclic graph with each group a vertex and each edge a parent/child relation between the groups such that the edge is directed to the parent. All possible paths in the graph have the constraint that they must eventually lead to a special *min_group* that has no parents and no assigned attributes. Child groups inherit the attributes of their parent groups such that the child group's "effective" set of attributes (i.e. the set of attributes both directly assigned to the group and inherited from other groups) consists of the union of all parent groups' "effective" attributes with the attributes directly assigned to the child. An example user group graph is shown in Figure 5.2, in which directly assigned attributes are shown under each group name. In this example, the effective set of attributes for the *Gradstudents* group would be $\{\{employee\_level, \{1\}\}$, $\{student\_level, \{1,2\}\}$, $\{room\_access, \{MC8, MC10, MC355, MC342, MC325\}\}$ as the *employee_level* attribute is in-

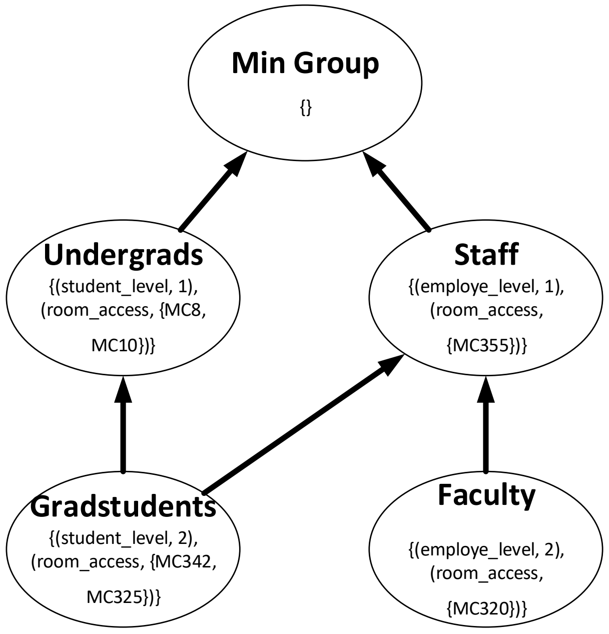

Min Group
{}
Undergrads
{(student_level, 1), (room_access, {MC8, MC10})}
Staff
{(employe_level, 1), (room_access, {MC355})}
Gradstudents
{(student_level, 2), (room_access, {MC342, MC325})}
Faculty
{(employe_level, 2), (room_access, {MC320})}

Figure 5.2: Example user group hierarchy represented as a graph. The large bold text denotes the group's name, beneath which the set of directly assigned attributes is shown.

Listing 5.1: Example HGABAC HGPLv1 Policy Permission Pairs

```
P1 = (user.age >= 18 AND object.title = "Adult␣Only␣Book", read)
P2 = (user.id = object.author, write)
P3 = (user.role IN {"doctor", "intern", "staff"} AND
      user.id != object.patient, read)
P4 = (object.type = "program" AND object.required_certifications
      SUBSET user.certifications, run)
```

herited from the *Staff* group and values of the other attributes are merged with the values from the parent groups (*Staff* and *Undergrads*). In the case of the *Faculty* group, the effective set of attributes would be $\{\{employee\_level, \{1,2\}\}, \{room\_access, \{MC355, MC320\}\}$, only inheriting attributes from the *Staff* group.

In addition to the core HGABAC model, an attribute-based policy language (HGPLv1) was created to support policy creation and evaluation. HGPLv1 represents policies as C style boolean statements that may evaluate to *TRUE*, *FALSE* or *UNDEFINED*. A resulting evaluation of *TRUE* implies that access should be granted, *FALSE* that it should be denied and *UNDEFINED* if the policy can not be properly evaluated at the current time (equivalent to a result of *FALSE* for access control decision purposes). Policies are associated with a set of operations that they grant if satisfied.

Listing 5.1 presents a number of example policies that are possible in HGPLv1. Policy *P1* states that any user with an age of 18 or older can read the book with the title "Adult Only Book". Policy *P2* allows a user to write to any object they are an author of. Policy *P3* limits access to read a medical record to users who have the role doctor, intern or staff but only if they are not listed as a patient in that record. And finally, policy *P4* specifies that a user can run a program if they have the required certifications listed in the program's certifications attribute.

It has been shown[Servos and Osborn 2014] (Chapter 3 Section 3.5) that HGPLv1 and HGABAC are capable of emulating MAC, DAC and hierarchical RBAC (though not separation of duties) and that their attribute groups result in less complex (in terms of the number of assignments and relations between access control entities) representations than standard (non-hierarchical) ABAC models under a number of hypothetical use cases.

5.3 Related Work

A number of generic access control frameworks and architectures exist that could conceivably be used to support HGABAC. The most notable of these are the Security Assertion Markup Language (SAML)[Hughes and Maler 2005], the eXtensible Access Control Markup Language (XACML)[Anderson et al. 2003] and the AAA Authorization Framework[Vollbrecht et al. 2000]. SAML provides an XML-based standard for exchanging authentication and authorization information commonly used for Single Sign-On (SSO). XACML provides a XML-based standard for representing and sharing access control policies and an accompanying architecture that follows the AAA Authorization Framework which describes an authorization framework as a combination of the following distributed policy modules:

- **Policy Retrieval Point (PRP)**: Point at which a policy is retrieved from a Policy Repository.
- **Policy Decision Point (PDP)**: Point where a policy from a PRP is evaluated based on information from one or more PIPs.
- **Policy Enforcement Point (PEP)**: Point at which the policy is enforced (i.e. the point at which a user is denied or granted access to a service based on the result of the PDP).
- **Policy Information Point (PIP)**: Point at which information needed to evaluate a policy is retrieved. In the case of ABAC, this is normally attributes and their values.
- **Policy Administration Point (PAP)**: Point at which policies are administered and/or created[1].

While it may be possible to create a HGABAC architecture-based system solely on these standards (and this is a possible direction for future work mentioned in Section 5.6), such a solution would face a number of issues and shortcomings addressed by our HGAA architecture:

- **Off-Line PIPs**: In the most common use of the XACML architecture, a PDP requests the policy information required to evaluate a policy from a PIP after it receives a request from a PEP (presumably triggered by a user attempting to access a service). However, this approach is problematic if PIPs are unavailable (e.g. a user's home domain may not have a PIP for user attributes that is available continuously or may wish to only assign attributes off-line) or such requests introduced excessive overhead (e.g. if a large number of PIPs need to

[1]Not defined in the AAA Authorization Framework, only in XACML

be contacted to collect a full set of a user's attributes). HGAA provides a solution in which no outside PIPs need be contacted if an attribute certificate is available.

- **Public Key Infrastructure (PKI) Overhead**: X.509 Attribute Certificates[Farrell et al. 2010] and SAML require the use of X.509 PKI which may be unavailable or overkill in many cases. Additionally, requiring attribute certificates to be accompanied by the holder's Public Key Certificate can weaken anonymity, a key feature of ABAC. HGAA overcomes this by using a simplified public key based trust system between Policy and Attribute Authorities and specifying an attribute certificate format that does not require X.509 Public Key Certificates and allows for pseudonymity.

- **Future Support for Delegation**: A future goal of HGABAC is to support one or more models of delegation as described in [Servos and Osborn 2016] (Chapter 4). While XACML does have a delegation profile[Rissanen et al. 2009], the style of delegation supported is closer to administration than traditional user-to-user delegation. HGAA provides a number of points upon which future extensions enabling a variety of delegation and revocation models can be built while maintaining backwards compatibility.

- **HGABAC Specific vs. Generic Architecture**: XACML supports a large range of flexible access control policies that do not necessarily conform to the HGABAC model or policy language. Use of XACML would require the creation of a new XACML HGABAC profile and means of translating HGABAC policies into XACML. As HGAA is designed specifically for HGABAC, native HGABAC policies are supported without translation to a secondary policy language and no features are restricted or compromised while ensuring that the HGABAC model is enforced.

- **Lightweight Approach**: XACML and other attribute-based architectures provide flexibility and interoperability at the cost of increasing complexity and verbosity of access control policies (for example see the comparison of a XACML and HGABAC policy in Listings 5.2 and 5.3) as well as the supporting infrastructure. HGABAC and HGAA take a lightweight approach, aiming to provide a simplified yet powerful policy language that requires minimal infrastructure to support.

In addition to the generic standards mentioned, the work by S. Bhatt et al.[Bhatt et al. 2017] towards creating an authorization architecture and implementation of HGABAC utilizing the NIST Policy Machine (PM)[Ferraiolo et al. 2014, 2011] is also of note. S. Bhatt et al. introduce a restricted HGABAC model (rHGABAC) formalized as a single-value enumerated policy enabling it to be implemented using a bare minimum version of the PM to take advantage of the PM's existing access control framework. While they were successful in implementing the core features of HGABAC, the restricted model only allowed for simplified policies and lacks all assignment relations from the original HGABAC model. Further, the authorization architecture presented is limited to a single PIP that is combined with the PDP and PAP.

Listing 5.2: XACML rule to only allow access between 9AM and 5PM.

```
<Rule RuleId="TimeRule" Effect="Permit">
  <xacml3:Description>Allow if time between 9AM and 5PM</xacml3:Description>
  <xacml3:Target/>
  <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeSelector
                   DataType="http://www.w3.org/2001/XMLSchema#time"
                   AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
          09:00:00
      </AttributeValue>
    </Apply>
    <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
        <EnvironmentAttributeSelector
                   DataType="http://www.w3.org/2001/XMLSchema#time"
                   AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
      </Apply>
      <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">
          17:00:00
      </AttributeValue>
    </Apply>
  </Condition>
</Rule>
```

Listing 5.3: HGABC Policy to only allow access between 9AM and 5PM.

```
    env.time_of_day_hour >= 9 AND env.time_of_day_hour <= 17
```

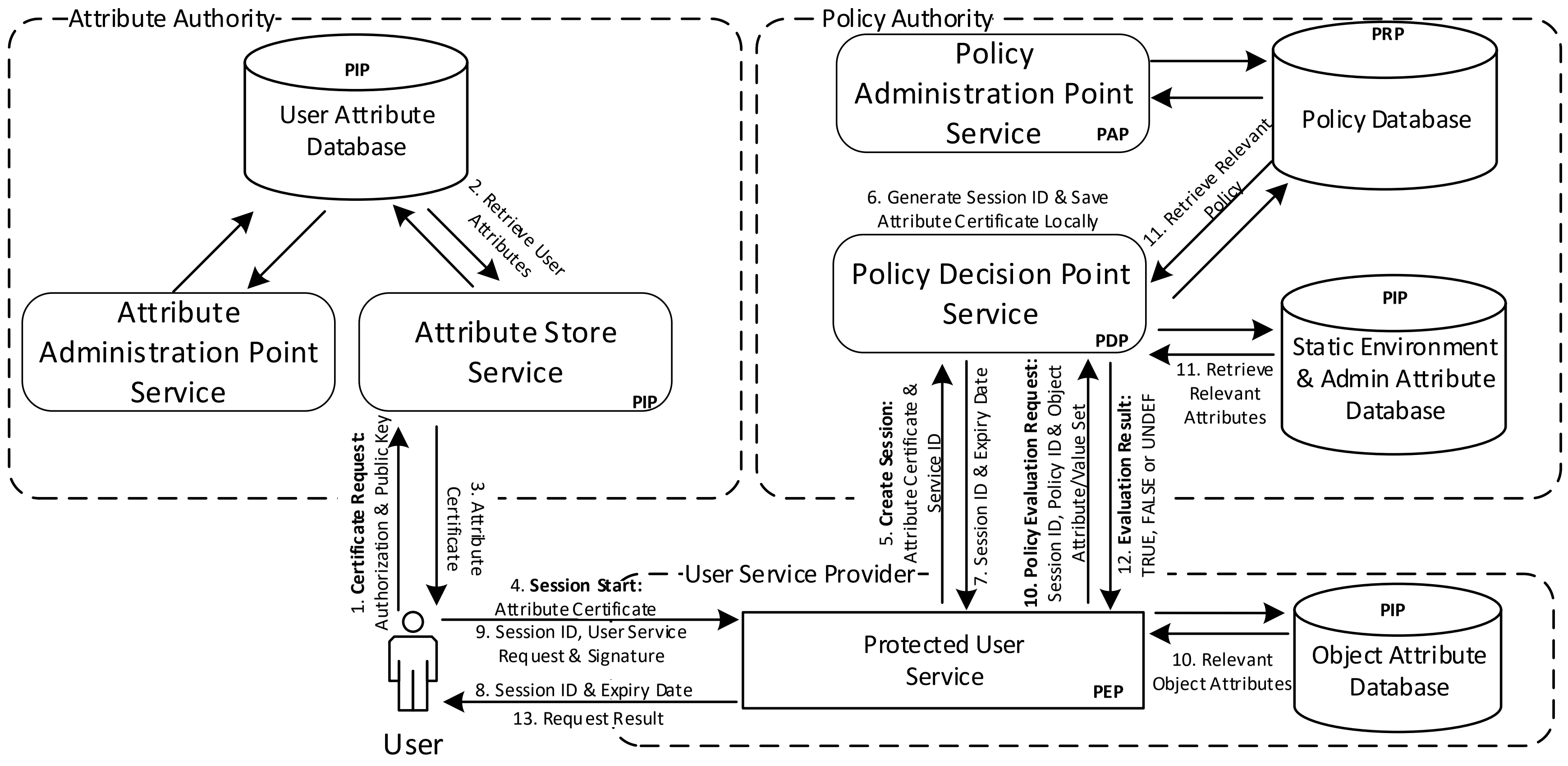

Attribute Authority
PIP
User Attribute Database
2. Retrieve User Attributes
Attribute Administration Point Service
Attribute Store Service
PIP
1. Certificate Request: Authorization & Public Key
3. Attribute Certificate
Policy Authority
Policy Administration Point Service
PAP
PRP
Policy Database
6. Generate Session ID & Save Attribute Certificate Locally
11. Retrieve Relevant Policy
Policy Decision Point Service
PDP
PIP
Static Environment & Admin Attribute Database
11. Retrieve Relevant Attributes
5. Create Session: Attribute Certificate & Service ID
7. Session ID & Expiry Date
10. Policy Evaluation Request: Session ID, Policy ID & Object Attribute/Value Set
12. Evaluation Result: TRUE, FALSE or UNDEF
User Service Provider
4. Session Start: Attribute Certificate
9. Session ID, User Service Request & Signature
8. Session ID & Expiry Date
13. Request Result
User
Protected User Service
PEP
10. Relevant Object Attributes
PIP
Object Attribute Database

Figure 5.3: HGAA services and information flow. Numbers indicate order of requests. Dotted lines denote components of a service (i.e. Attribute Store Service is a component/subservice of the Attribute Authority Service).

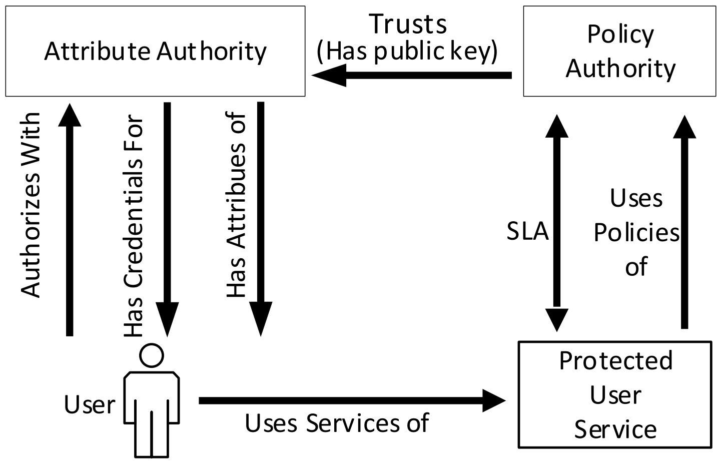


Figure 5.4: Relationships between HGAA services and users.

5.4 Architecture

The HGAA architecture is comprised of three core service types; the Attribute Authority service (discussed in Section 5.4.2), the Policy Authority service (discussed in Section 5.4.5) and the services provided by the User Service Provider (referred to as "User Services" and discussed in Section 5.4.4). Using terminology from the AAA Authorization Framework, the Attribute Authority would be analogous to a PIP, the Policy Authority to a combined PDP, PRP and PAP and the user services to a PEP. A simplified (only showing a single instance of each service and one user) representation of this architecture and the information flow between the services is shown in Figure 5.3.

Many instances of Attribute Authorities, Policy Authorities and User Services are allowed to coexist across diverse security domains. Interoperability between these services and domains is possible so long as a trust relation has been established at a prior time between a given Attribute Authority and a given Policy Authority. From a technical standpoint, this trust relation consists of a Policy Authority listing an Attribute Authority's public key as trusted (thus accepting attribute certificates issued by this authority). From a practical standpoint, this means that users who are issued an attribute certificate (hereinafter referred to as an "AC" and discussed in more depth in Section 5.4.3) from one domain/organization can access the services of another domain/organization if a trust relation exists between their Attribute and Policy Authorities and the user satisfies the required policies. User Services are able to utilize HGABAC by employing the services provided by a Policy Authority to evaluate user requests (which include an AC issued by a trusted Attribute Authority). These relations between services and users are shown in Figure 5.4.

The following subsections describe each major HGAA component in more depth including the requests between services shown in Figure 5.3.

5.4.1 HGABAC Namespace

As multiple Attribute Authorities may exist and do not directly communicate, attributes from different sources must be uniquely identifiable and conflicts avoided. This issue is not isolated

Listing 5.4: HGABAC URI Based Namespace Grammar

```
Absolute URI:
     hgabac://<authority>[[/<type>]/<element_name>]

Relative URI:
     [/]<type>/<element_name>
     | [/]<element_name>

type:
     user
     | group[/user | /object]
     | attribute[/<att_sub_type>]
     | object[/<obj_sub_type>]
     | session
     | operation
     | permission
     | policy
     | service

att_sub_types:
     user
     | object
     | environment
     | admin
     | connection
     | unknown

obj_sub_types:
     file
     | service
     | hardware
     | program
     | database
     | meta
     | unknown

element_name:
     defined by regex: [a-zA-Z0-9\.\-\_]+

authority:
     <host>[:<port>]

host:
     valid hostname as per RFC 1123

port:
     defined by regex: [1-9][0-9]*
     must be less then 65536.
```

to HGAA but an open ABAC problem that has been identified in recent literature[Servos and Osborn 2017; Hu et al. 2013] (and discussed in Chapter 2 Section 2.4.7). We offer a partial solution that is similar to the scheme used in XACML, in which each attribute (and all other HGABAC elements) is given a unique URI[Berners-Lee et al. 2005] based on the grammar given in Listing 5.4. The key difference between our namespace scheme and XACML's is our support for and treatment of relative URIs.

Absolute URIs such as *hgabac://cs1.ca/attribute/user/age* specify a HGABAC element (in this case a user attribute) from a specific authority (in this case *cs1.ca*). A relative URI such as */attribute/user/age* would specify a HGABAC element from any authority (in this case an user attribute named *age* from any authority). By omitting other parts of the path, relative URIs can make broader matches. For example, */attribute/role* would match any role attribute regardless of attribute type or issuing authority. Supporting both absolute and relative URIs in this fashion allows policies to be created that reference attributes from a distinct authority (absolute reference) or any attribute of the same name and type (relative reference).

5.4.2 Attribute Authority

The Attribute Authority provides services to administer the user group hierarchy and user attribute information in a given domain/organization and issue ACs to users within that domain. It is comprised of two subservices and a database (as shown in Figure 5.3); the Attribute Administration Point Service, the Attribute Store services, and the User Attribute Database. The Attribute Administration Point Service provides administrative operations related to managing and assigning user attributes. The User Attribute Database stores user attribute assignments for users in the Attribute Authority's domain. The Attribute Store Service provides two important functions; first and most importantly it deals with certificate requests and issuing ACs and secondly it maintains a revocation list for all ACs issued in the domain that have been revoked.

The certificate request process proceeds as follows (and is shown as steps 1, 2 and 3 in Figure 5.3):

1. The user sends a certificate request to the Attribute Authority containing a list of user attributes they wish to activate for a given session, their public key from a public/private key pair generated solely for this session and their credentials for authenticating with the Attribute Authority. The method of authentication is left as an implementation detail for the Attribute Authority and may be domain specific. As per the HGABAC specification, users may activate a subset of their assigned and inherited user attributes to allow for principle of least privilege and prevent identifying information being included in unneeded attributes.

2. If the user's credentials are valid, the Attribute Authority requests the activated attributes assigned to the user and their values from the Attribute Database and generates an AC as described in Section 5.4.3. This AC contains the public key provided by the user in step 1, such that the user can provide proof of ownership of the AC using their corresponding private key without providing identifying information.

3. The generated AC is issued to the user who may now use it to authenticate with User Service Providers including those outside of the user's home domain without providing any

additional credentials. No further communication is required between the user and the Attribute Authority (or the Attribute Authority and any other component of the HGAA) for this session after the certificate has been issued and this process may be completed off-line.

Revocation of ACs may happen in two ways. First, the Attribute Authority may embed revocation rules within the AC that define conditions under which the certificate is valid. For example, a common revocation rule would be to revoke a certificate after a set date/time. The second means of revoking a certificate is through the revocation list maintained by the Attribute Store subservice. AC serial numbers listed in the list are considered to no longer be valid and revoked. Providing this list is optional and requires the Attribute Authority to be on-line and accessible (at least periodically) by Policy Authorities such that revocation lists can be synchronized.

5.4.3 Attribute Certificate

Attribute Certificates (ACs) allow users to offer proof of their attributes to User Service Providers. This proof comes in the form of a cryptographically signed certificate issued by a trusted Attribute Authority accompanied by the public/private session key pair generated by the user at the start of the session (the public key being embedded in the certificate and the user proving they are in possession of the corresponding private key by signing request).

We loosely base the design of our AC format on X.509 Attribute Certificates[Farrell et al. 2010], due to their simple and lightweight design and use in related literature for RBAC[Chadwick et al. 2003], but without the need for Public Key Certificates, supporting PKI or binding with a user's identity. The logical format of the certificate is defined using Abstract Syntax Notation One (ASN.1)[Recommendation 2015] in Listing 5.5. Our AC consists of the following 8 sections:

- **ACInformation:** Contains meta information about the AC. Consists of the certificate version number (to facilitate compatibility with future versions), a globally unique serial number[2] and the date and time the certificate was issued.

- **ACIssuer:** Identifying information about the issuer of the AC. Comprised of the public key of the issuer (the attribute authority) including information about the public key algorithm used, a unique identifier for the issuing authority formatted as a HGABAC Namespace URI (e.g. *hgabac://cs1.ca* for the authority *cs1.ca*), and optionally, a common descriptive name for the issuing authority and a URL to the authorities web service if publicly available.

- **ACHolder:** Pseudonymous information about the holder of the AC (i.e. the user the AC was issued to). Contains the public key the user is using for this session including information about the public key algorithm used, a unique identifier for the user formatted as a HGABAC Namespace URI[3] (e.g. *hgabac://cs1.ca/user/u1135* for the authoirty *cs1.ca* and the user *u1135*) and optionally a common descriptive name for the user if anonymity is not desired.

[2] Only needs to be globally unique for practical purposes and can be based on probability rather than requiring coordination between issuing authorities.

[3] Note that this identifier is a pseudonym for the user and should not contain real identifying information. This identifier can be different or partially randomized for each session.

Listing 5.5: AC Format Defined in ASN.1

```
AttributeCertificate ::= SEQUENCE {
     information    ACInformation,
     issuer         ACIssuer,
     holder         ACHolder,
     attributes     SEQUENCE OF Attribute,
     revocation     RevocationRules,
     delegation     DelegationRules OPTIONAL,
     extensions     SEQUENCE OF ACExtensions OPTIONAL,
     signature      ACSignature
}

ACInformation ::= SEQUENCE {
     version   ACVersion,
     serial    INTEGER,
     issued    DATE-TIME
}

ACVersion ::= INTEGER { v1(0) }

ACIssuer ::= SEQUENCE {
     issuerPublicKey          BIT STRING,
     issuerKeyAlgorithm       AlgorithmIdentifier,
     issuerUniqueIdentifier   OBJECT IDENTIFIER,
     issuerName               VisibleString OPTIONAL,
     issuerServiceURL         UTF8String OPTIONAL
}

ACHolder ::= SEQUENCE {
     holderPublicKey          BIT STRING,
     holderKeyAlgorithm       AlgorithmIdentifier,
     holderUniqueIdentifier   VisibleString,
     holderName               VisibleString OPTIONAL
}

Attribute ::= SEQUENCE {
     attributeID    OBJECT IDENTIFIER,
     attributeType  OBJECT IDENTIFIER,
     attributeValue ANY DEFINED BY attributeType OPTIONAL,
     attributeName  VisibleString OPTIONAL,
     ...
}

RevocationRules ::= SEQUENCE {
     validAfter               DATE-TIME,
     validBefore              DATE-TIME,
     revocationServiceURL     UTF8String OPTIONAL,
     ...
}

DelegationRules ::= SEQUENCE {
     ...
}

ACExtension ::= SEQUENCE {
     extensionID    OBJECT IDENTIFIER,
     ...
}

ACSignature ::= SEQUENCE {
     signatureAlgorithm  AlgorithmIdentifier,
     signatureValue      BIT STRING
}

-- As Defined in RFC 5280
AlgorithmIdentifier  ::=  SEQUENCE  {
     algorithm      OBJECT IDENTIFIER,
     parameters     ANY DEFINED BY algorithm OPTIONAL
}
```

- **Attribute Set:** The set of user attributes and their values that the user wishes to activate. Each attribute in the set must be assigned to the user directly or inherited through group membership as described in HGABAC. Each attribute contains a unique identifier following the HGABAC Namespace (e.g. *hgabac://cs1.ca/attribute/user/department*), the attribute's type, value and optionally a common descriptive name for the attribute (e.g. "Department Name"). In addition to the requested user attributes, this set also includes connection attributes that contain meta information about the AC its self (such as date/time issued, issuer UID, holder UID, group information, etc.) such that these properties can be used in HGABAC policies.

- **ACRevocationRules:** Set of rules under which an AC is valid. For the first version of the AC presented in this work, these are limited to valid before and valid after rules. However, space is left for extension and future work. Optionally, a URL to a revocation list web service may be given. Such a service would provide a list of all revoked AC serial numbers issued by the authority.

- **ACDelegationRules:** A place holder for delegation rules to be added in future extensions. It is envisioned that this will allow for Attribute Delegation as described in Chapter 4 (published in [Servos and Osborn 2016]).

- **Extension Set:** A place holder for miscellaneous future extensions. The only requirement given is that extension must have a unique identifier.

- **ACSignature:** A cryptographic signature of all other sections of the attribute certificate using the issuing authority's private key (that corresponds to the public key given in the ACIssuer section). Also included is information about the signature algorithm used. ACs are signed by the attribute authority when issued to users as part of a certificate request and offer proof of the authority trusting that the attributes contained describe the holder of the certificate.

We support both a human readable text-based encoding of an attribute certificate as well as a more efficient and less ambiguous byte encoding detailed in Figures 5.5 and 5.6. An example of an attribute certificate in the text-based encoding is shown in Listing 5.6.

Figure 5.5: Attribute certificate byte encoding

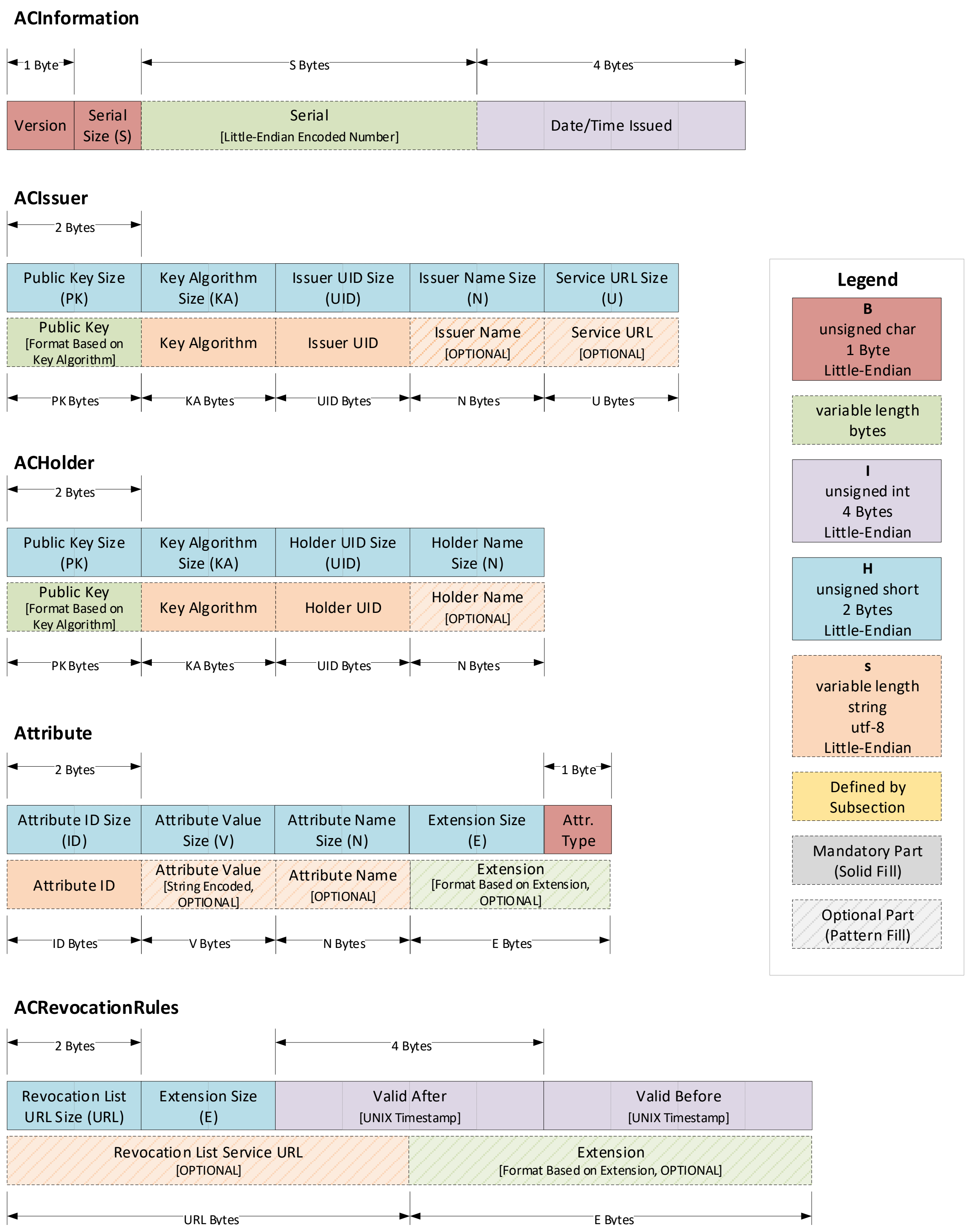
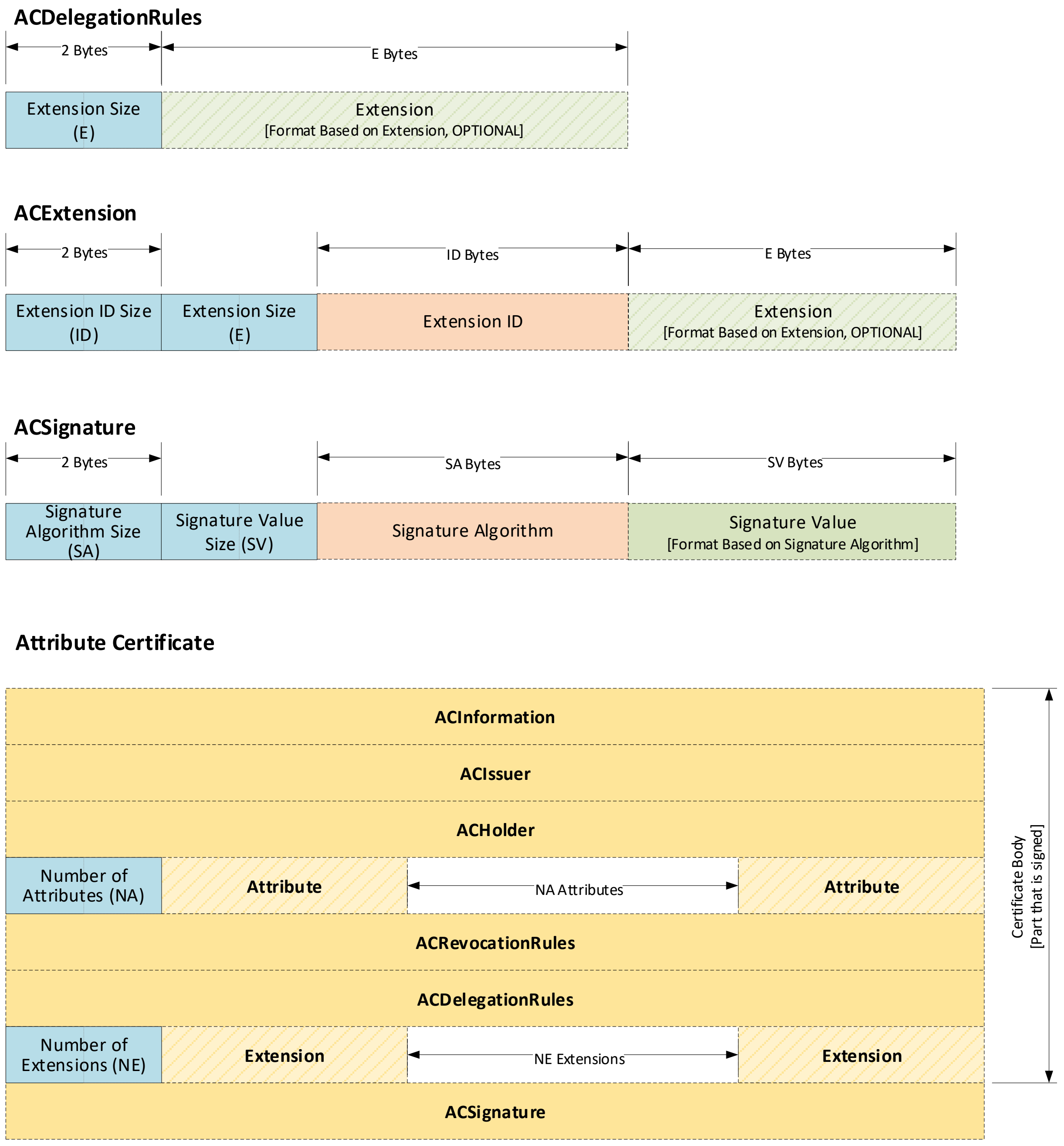


Figure 5.6: Attribute certificate byte encoding (continued from Figure 5.5)



Listing 5.6: Example AC in Text-Based Encoding. Keys, signatures and attribute list are abbreviated.

```
---- BEGIN ATTRIBUTE CERTIFICATE ----
FORMAT: TEXT
VERSION: 1
==== BEGIN INFORMATION ====
VERSION: 1
SERIAL: 1458702832854692305562335215823881962486460489003
ISSUED: 1513313064
==== END INFORMATION ====
==== BEGIN ISSUER ====
PUBLIC KEY: LS0tLS1CR...ZLS0tLS0=
KEY ALGORITHM: RSA[2048]
UID: hgabac://cs1.ca
NAME: CS1.CA Attribute Authority
URL: http://cs1.ca/AttributeAuthority/
==== END ISSUER ====
==== BEGIN HOLDER ====
PUBLIC KEY: LS0tLS1CRU...VZLS0tLS0=
KEY ALGORITHM: RSA[2048]
UID: hgabac://cs1.ca/user/u1135
NAME: Daniel Servos
==== END HOLDER ====
==== BEGIN ATTRIBUTE SET ====
#### BEGIN ATTRIBUTE: /attribute/user/account_balance ####
ATTRIBUTE ID: /attribute/user/account_balance
ATTRIBUTE TYPE: AttributeType.FLOAT
ATTRIBUTE VALUE: 9999.9999
ATTRIBUTE NAME: account_balance
#### END ATTRIBUTE: /attribute/user/account_balance ####
#### BEGIN ATTRIBUTE: /attribute/user/age ####
ATTRIBUTE ID: /attribute/user/age
ATTRIBUTE TYPE: AttributeType.INT
ATTRIBUTE VALUE: 31
ATTRIBUTE NAME: age
#### END ATTRIBUTE: /attribute/user/age ####
#### BEGIN ATTRIBUTE: /attribute/user/admin ####
ATTRIBUTE ID: /attribute/user/admin
ATTRIBUTE TYPE: AttributeType.BOOL
ATTRIBUTE VALUE: TRUE
ATTRIBUTE NAME: admin
#### END ATTRIBUTE: /attribute/user/admin ####
#### BEGIN ATTRIBUTE: /attribute/user/courses ####
ATTRIBUTE ID: /attribute/user/courses
ATTRIBUTE TYPE: AttributeType.SET
ATTRIBUTE VALUE: CS2211,CS2034,CS1234,CS5678,CS9000
ATTRIBUTE NAME: courses
#### END ATTRIBUTE: /attribute/user/courses ####
#### BEGIN ATTRIBUTE: /attribute/connection/ac_version ####
ATTRIBUTE ID: /attribute/connection/ac_version
ATTRIBUTE TYPE: AttributeType.INT
ATTRIBUTE VALUE: 1
ATTRIBUTE NAME: ac_version
#### END ATTRIBUTE: /attribute/connection/ac_version ####
...many attributes omitted for length reasons...
#### BEGIN ATTRIBUTE: /attribute/connection/aauth_uid ####
ATTRIBUTE ID: /attribute/connection/aauth_uid
ATTRIBUTE TYPE: AttributeType.STRING
ATTRIBUTE VALUE: hgabac://cs1.ca
ATTRIBUTE NAME: aauth_uid
#### END ATTRIBUTE: /attribute/connection/aauth_uid ####
==== END ATTRIBUTE SET ====
==== BEGIN REVOCATION RULES ====
VALID AFTER: 1513313064
VALID BEFORE: 1513316664
URL: http://cs1.ca/AttributeAuthority/revocation_list
==== END REVOCATION RULES ====
==== BEGIN SIGNATURE ====
SIGNATURE ALGORITHM: RSASSA-PKCS1-v1_5:SHA256
SIGNATURE VALUE: j6Zk7zl...e/eX1nGQ==
==== END SIGNATURE ====
---- END ATTRIBUTE CERTIFICATE ----
```

5.4.4 User Service Provider

The User Service Provider provides services to end users that they wish to protect using HGABAC and maintain the object group hierarchy and object attribute information relevant to their services. Providers designate the policies under which their services may be accessed but outsource the work of storing and evaluating these policies to the Policy Authority. Requests upon User Services are allowed/denied based on the Policy Authority's evaluation of access policies and the attributes contained in the AC as well as the attributes of objects the service will be accessing and the current value of environment, connection and administrative attributes (as described in HGABAC).

Creation of a session between the user and User Service are handled as follows (shown as steps 4, 5, 7 and 8 in Figure 5.3):

4. A user starts a session with a User Service by sending their AC (issued to the user by the Attribute Authority in steps 1-3) to the User Service.

5. The User Service creates a session for the user by forwarding the user's AC to the Policy Authority in a Create Session request. This request contains the user's AC and the User Service's unique ID as well as authentication information for the User Service, if authentication between the Policy Authority and User Service is required (depends on implementation).

7. The Policy Authority responds with session information including a unique session ID and expiry date/time. The User Service saves a copy of the AC and session information until the expiry date/time. For the remainder of the session it is no longer required for the user or User Service to transmit the AC.

8. The User service responds to the user's session request with the session ID and expiry date/time from the Policy Authority. A session is considered active and valid so long as the AC is not revoked, the session expiry date/time is not past and the user has the session ID and private key corresponding to the public key embedded in the AC. A user may terminate a session by destroying the session ID and/or private key (destroying the private key would terminate all sessions associated with the AC).

Once a session has been established, a user may make requests upon the User Service as follows (shown as steps 9, 10, 12 and 13 in Figure 5.3):

9. The user makes a request on the User Service that includes the session ID received in step 8 and signs this request with their private key matching the public key in the AC.

10. The User Service validates the signature on the request (using the public key in the user's AC) and if valid, sends a Policy Evaluation request to the Policy Authority which includes the session ID, the policy ID associated with the HGABAC policy that must be passed for the request to be allowed and the set of relevant object attributes (and their values).

12. The Policy Authority responds to the Policy Evaluation request with either *TRUE*, *FALSE*, or *UNDEF* based on the ternary logic used in HGABAC. A *TRUE* response indicates that the policy has been satisfied and the user's request should be allowed. A response of *FALSE*

indicates that the policy has not been satisfied and the request should be denied. A *UNDEF* response indicates that the policy could not be evaluated (e.g. an attribute in the policy was not available) and the request should be denied.

13. Based on the result of the Policy Evaluation request, the User Service either fulfills the user's request and replies with an appropriate response or responds with a message indicating that the request was denied (optionally with more details as to why).

Further requests may be made by the user in a like manner without the need to create a new session so long as the session is not expired or terminated.

5.4.5 Policy Authority

Policy Authorities store, manage and evaluate HGABAC policies on behalf of User Service Providers. Policies are expressed in the Hierarchical Group Policy Language Version 2 (HGPLv2), an updated version of the policy language introduced in the original HGABAC work (from Chapter 3 and [Servos and Osborn 2014]). The grammar for the updated language is given in Listing 5.7 in Augmented Backus–Naur Form (ABNF)[Crocker and Overell 1997]. The most notable changes are the use of HGABAC Namespace URIs in place of attribute names and the addition of policy references. Policy references allow policies to reference other policies such that policies can be combined using basic logical operations (*AND*, *OR* and *NOT*). For example, if *P1* = `/user/age >= 18 OR /user/parent_consent` and *P2* = `/object/author = /user/id` then a third policy could be created that references *P1* and *P2*, *P3* = `/policy/P1 AND NOT /policy/P2` that would be equivalent to *P3* = `( /object/author = /user/id ) AND NOT ( /object/author = /user/id )`. Policies are restricted from creating recursive or circular references and references to unavailable policies result in *UNDEF*.

The Policy Authority Service is comprised of two subservices, the Policy Administration Point Service and the Policy Decision Point Service, and two databases, the Policy Database and the Environment & Administrative Attribute Database. The Policy Administration Point Service allows for the creation and management of policies by User Service Providers as well as the management of administrative attributes (as defined in HGABAC). The Policy Database stores HGPLv2 policies available for use by User Services and the Environment & Administrative Attribute Database stores the current values of environment and administrative attributes. The Policy Decision Point Service authenticates user AC and evaluates them against stored policies on behalf of a User Service.

Policy Authorities maintain a list of trusted Attribute Authorities along with their unique ID and public key. AC are considered valid by a Policy Authority if they satisfy the following requirements:

- The issuer of the AC is in the Policy Authority's list of trusted Attribute Authorities.
- The issuer's public key and ID match those recorded by the Policy Authority.
- The issue date of the AC is not in the future and is inside of the valid before and after range given in the revocation rules.

Listing 5.7: HGPLv2 Grammar in ABNF. An update to the HGPLv1 grammar.

```
policy    = policy "OR" term / term
term      = term "AND" exp / exp
exp       = var op var / [ "NOT" ] bool_var
          / [ "NOT" ] "(" policy ")" / [ "NOT" ] policy_id
var       = const / att_id
bool_var  = boolean / att_id
op        = ">" / "<" / "=" / ">=" / "<=" / "!=" / "IN"
          / "SUBSET"
atomic    = int / float / string / "NULL" / boolean
const     = atomic / set
boolean   = "TRUE" / "FALSE" / "UNDEF"
set       = "{" "}" / "{" setval "}"
setval    = atomic / atomic "," setval
int       = [ "-" ] +( DIGIT )
float     = int "." +( DIGIT )
string    = DQUOTE *( %x20-21 / %x23-5B / %x5D-7E
                    / %x5C DQUOTE / %x5C %x5C ) DQUOTE
att_id    = <ATTRIBUTE URI FROM HGABAC NAMESPACE>
policy_id = <POLICY URI FROM HGABAC NAMESPACE>
```

- All revocation rules are met including the current time/date being within the valid before and after range.
- The serial of the AC is not listed in the Attribute Authorities revocation list (if available).
- The version of the AC and all extensions are compatible with the Policy Authority.
- The signature is valid and verifiable with the Attribute Authority's public key.

After a session has been established and the AC authenticated (as described in Section 5.4.4), the User Service evaluates a policy with the Policy Decision Point Service as follows (and shown in steps 10, 11 and 12 of Figure 5.3):

10. The User Service sends a Policy Evaluation request that includes the session ID, the policy ID, and the set of object attribute value pairs. The Policy Decision Point Service extracts the user and connection attributes from the user's AC (which it received when the session was created) and checks that the session and AC remain valid and no revocation rules have been triggered.
11. If the AC remains valid, the decision point requests the policy matching the given policy ID from the Policy Database. If the policy references any other policy, these policies are also requested and combined. The combined policy is analysed and needed environment and administrative attributes are requested from the Environment & Administrative Attribute Database.
12. The combined policy is evaluated and the result (*TRUE*, *FALSE* or *UNDEF*) is issued to the User Service.

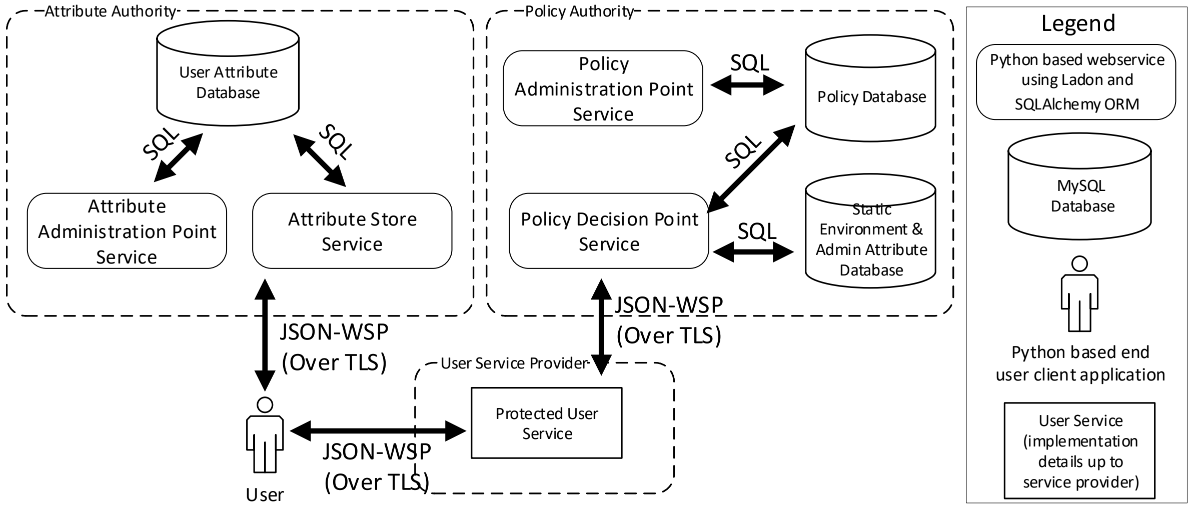

Attribute Authority
User Attribute Database
SQL
SQL
Attribute Administration Point Service
Attribute Store Service
Policy Authority
Policy Administration Point Service
SQL
Policy Database
SQL
Policy Decision Point Service
SQL
Static Environment & Admin Attribute Database
JSON-WSP (Over TLS)
JSON-WSP (Over TLS)
User Service Provider
Protected User Service
JSON-WSP (Over TLS)
User
Legend
Python based webservice using Ladon and SQLAlchemy ORM
MySQL Database
Python based end user client application
User Service (implementation details up to service provider)

Figure 5.7: Technology and standards used in the HGAA implementation.

5.5 Implementation & Preliminary Results

As shown in Figure 5.7, we implement each HGAA service as a Python-based JSON web service over TLS[4]. Web services are created with the Ladon[Simon-Gaarde] framework. HGAA databases are implemented using MySQL and the SQLAlchemy[SQLAlchemy] Object Relational Mapper (ORM) is used to facilitate communication between services and databases. Administrative services were not implemented at this time as our current focus is on evaluating the performance and scalability of the authentication and authorization features of the architecture.

5.5.1 Attribute Certificate

We evaluate our AC scheme in terms of size and time required to generate and sign the AC based on the number of user attributes included in the certificate (number of connection attributes remained constant at 35). The result of these comparisons are shown in Figures 5.8 and 5.9. We find that the size of the AC grows linearly with the number of user attributes activated at a rate of approximately 36 bytes[5] per attribute (for byte encoding with single value integer attributes) and that the time to generate an AC also grows linearly with the number of user attributes.

[4]Note that while TLS was used in this implementation, it is not required by HGAA or appropriate in all HGAA applications. TLS adds overhead and requires some level of PKI (e.g. certificates would be needed for each service), negating some advantages of the AC format. When TLS is not used, extra additions to the HGAA protocol are needed if messages may be intercepted, altered or repeated by an attacker to prevent replay attacks or intercepting the content of requests. Details and formalization of these additions is left to future work.

[5]This amount would vary based on a number of factors including the length of the attribute UID, common name, type and number of values.

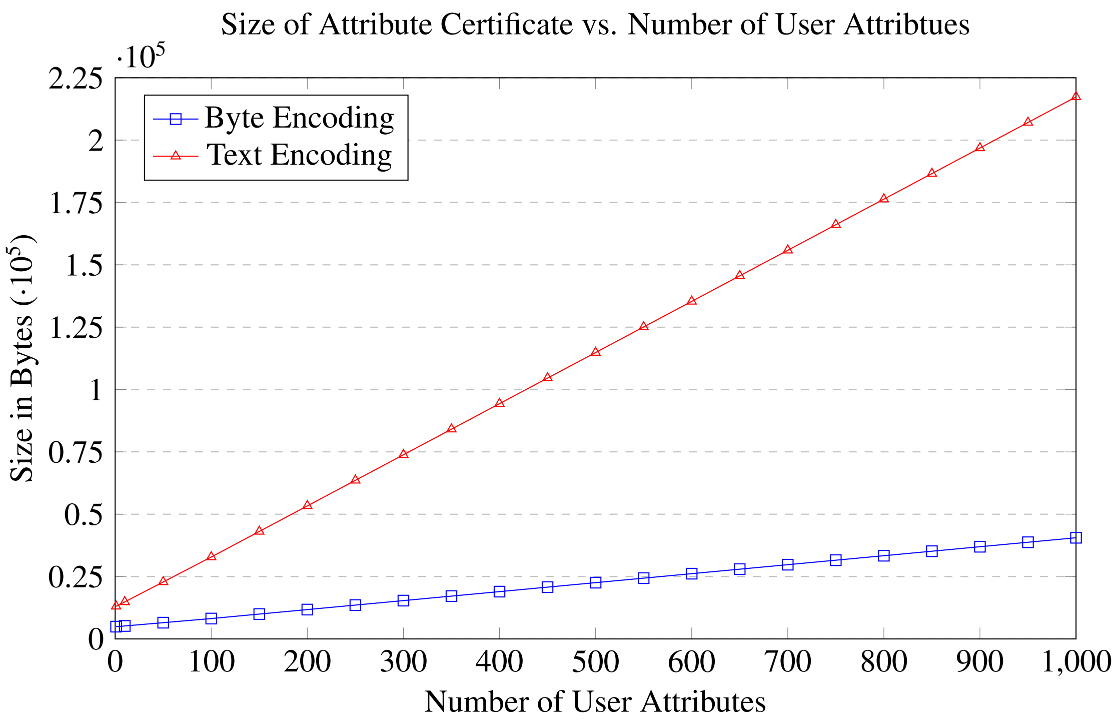


Figure 5.8: Size of Attribute Certificate vs. Number of User Attributes

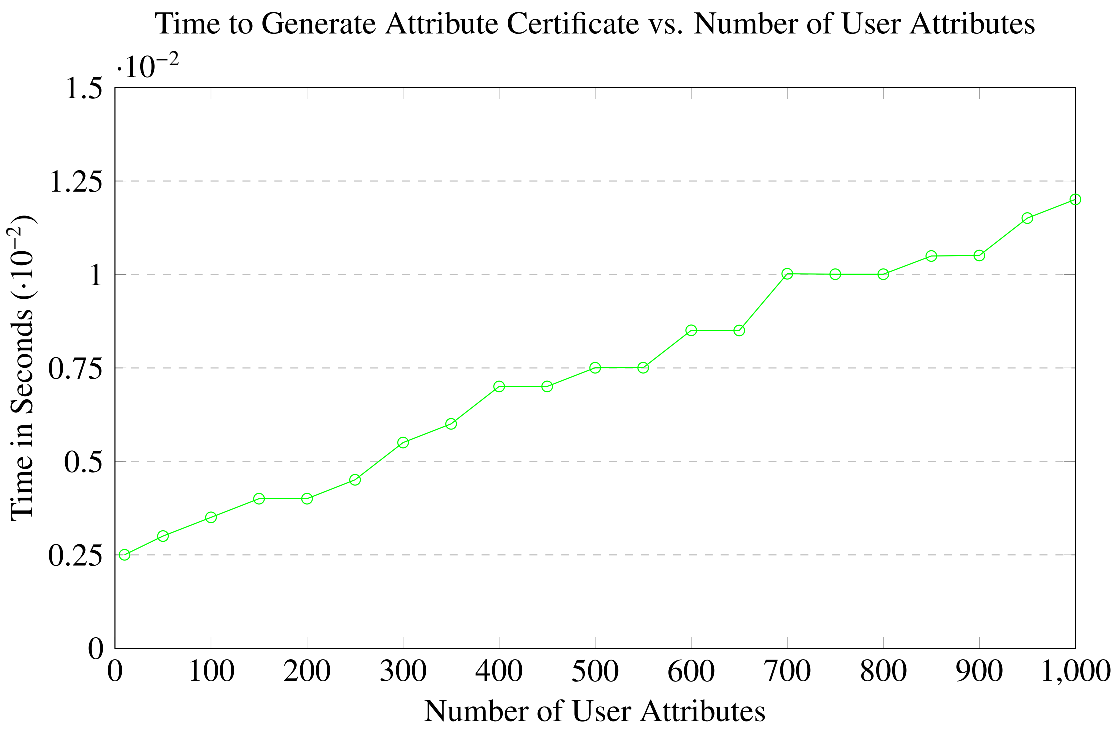


Figure 5.9: Time to Generate Attribute Certificate vs. Number of User Attributes

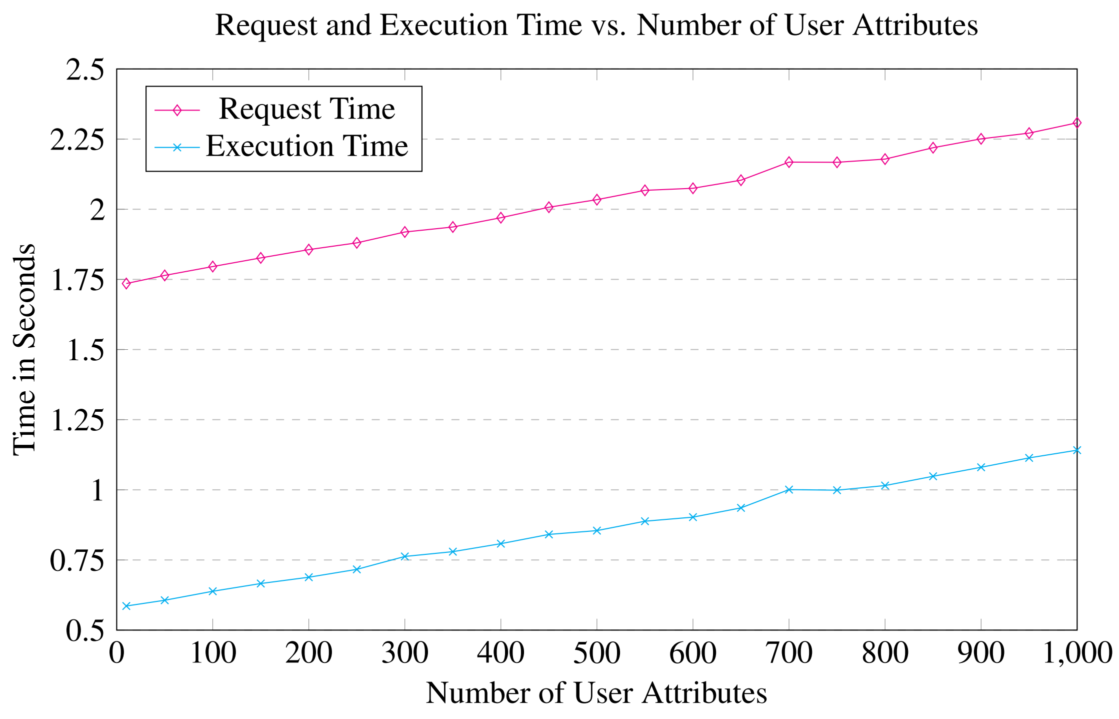


Figure 5.10: Attribute Store Service Request and Execution Time vs. Number of User Attributes

5.5.2 Attribute Authority

The Attribute Store Service of the Attribute Authority was evaluated in terms of request and execution time. The results of this evaluation are shown in Figure 5.10. Request time is defined as the time it takes a client to generate a Certificate Request, send it to the Attribute Store Service and receive a response (includes network and webserivce overhead), while execution time is defined as the time taken from the point the Attribute Store Service receives a request from a client to the point where the service issues a response (only includes time taken by the service to eventuate the request and generate a response). The number of connection attributes remained constant during testing (at 35).

We find that both the request and execution time increase linearly with the number of user attributes activated and included in the AC. The difference in request and execution time is related to network and web service overhead that we believe can be largely reduced by optimizing our implementation and moving to a different web service framework.

5.5.3 Policy Authority

An interpreter for the HGPLv2 policy language was implemented using python that utilizes a recursive descent parsing strategy. The interpreter is divided into a number of modules as shown in Figure 5.14. As an optimization step, the Abstract Syntax Tree (AST) of a given policy is computed and stored in a binary format in the Policy Database when a policy is added or modified. Additionally, an intermediate symbol table, listing the attributes referenced in the policy is computed and stored alongside the AST. Using these precomputed ASTs reduces the time required to fulfill Policy Evaluation requests from User Services and the intermediate

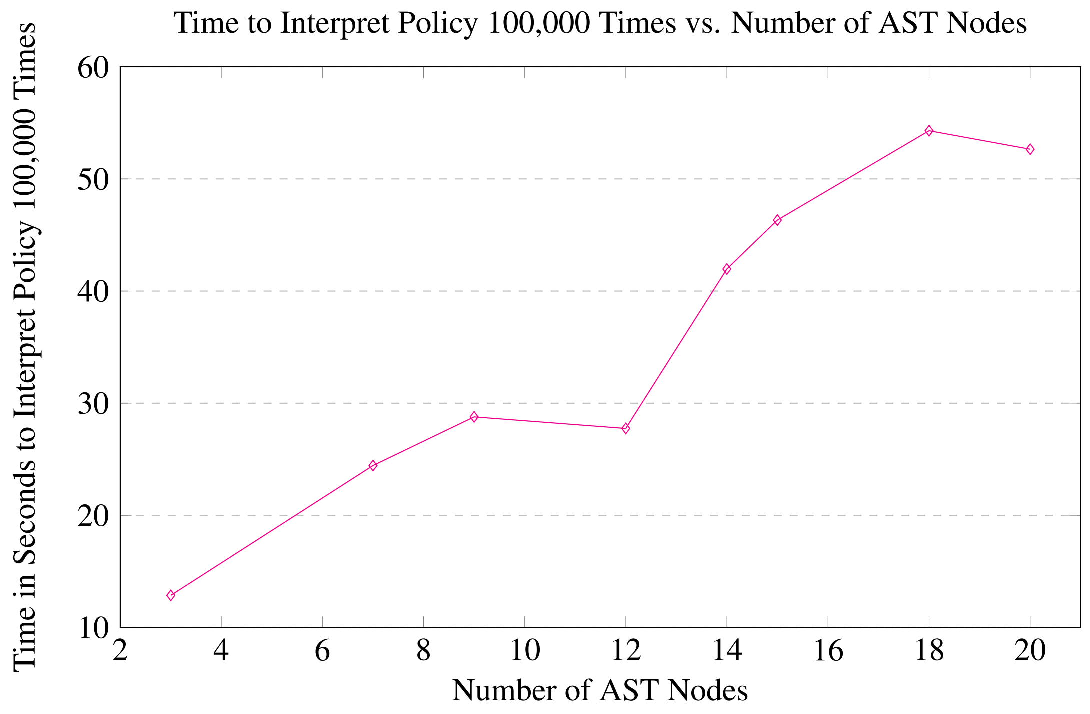


Figure 5.11: Time to Interpret Policy 100,000 Times vs. Number of AST Nodes. Policy is evaluated from scratch (not using precomputed AST).

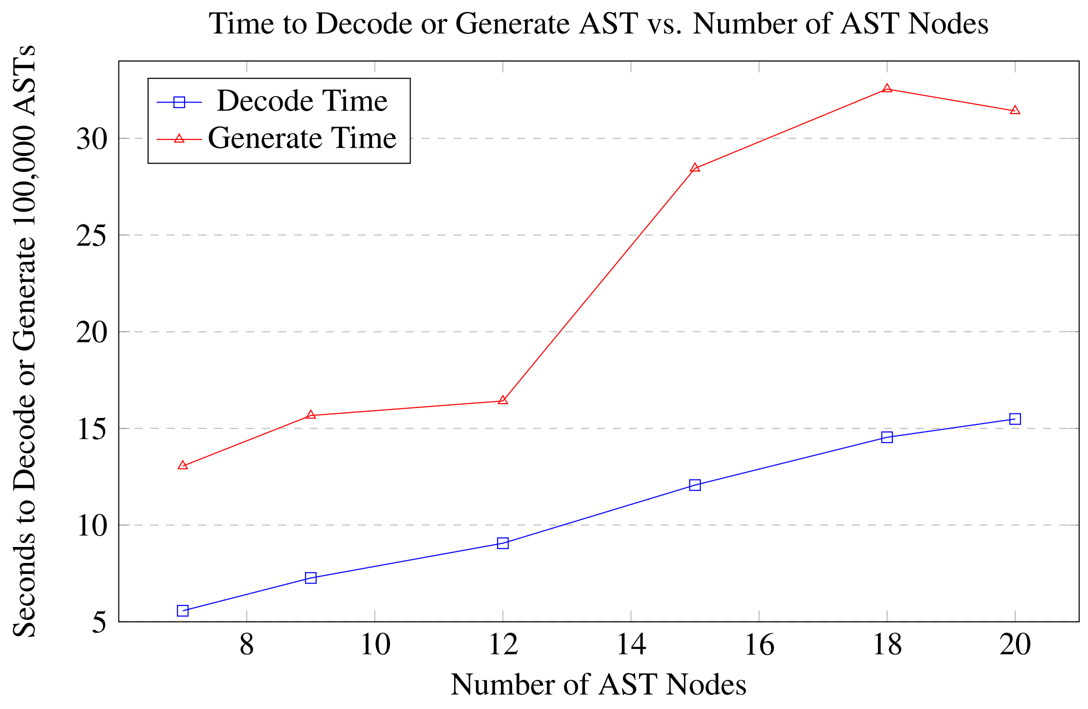


Figure 5.12: Time to Decode AST Byte Format or Generate AST from Scratch vs. Number of AST Nodes. Time given is to decode or generate AST 100,000 times.

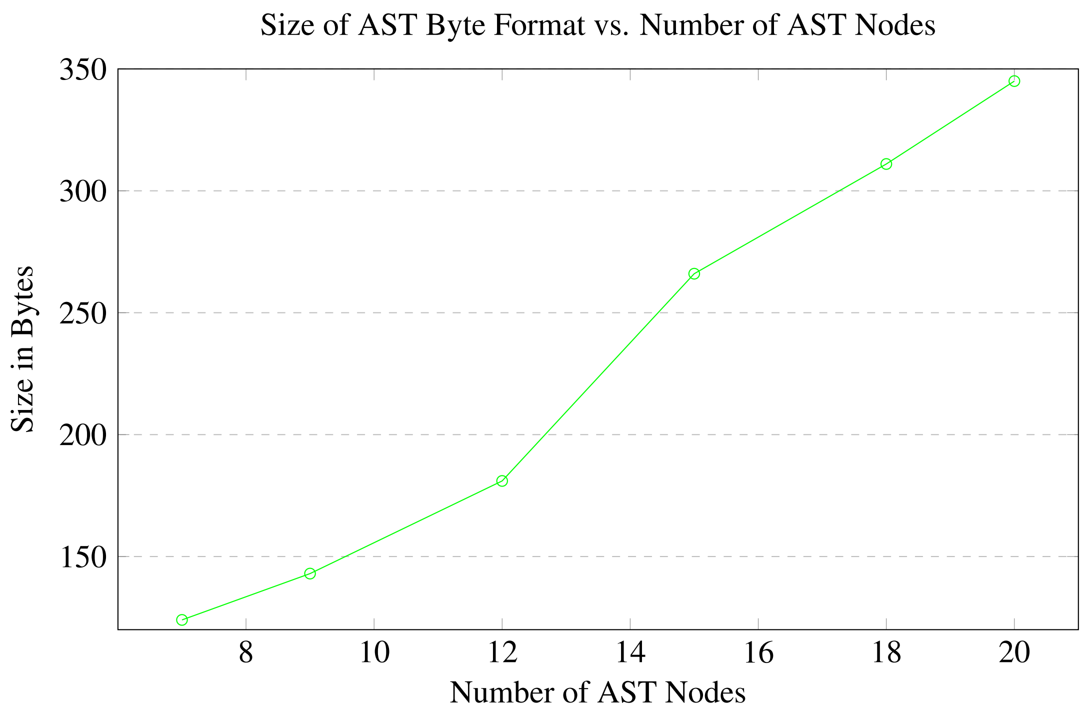


Figure 5.13: Size of AST Byte Format vs. Number of AST Nodes.

symbol table allows the Policy Decision Point Service to request required attributes without reanalysing the policy.

We evaluated the Policy Authority's performance handling Policy Evaluation requests based on the time required to interpret a policy, the time required to decode a precomputed AST and the size of the precomputed AST. Results of this analysis are shown in Figures 5.11 to 5.13. A linear relationship between the number of nodes in a policy's AST and the time required to evaluate the policy was found. Similarly, the time to generate or decode a policy AST also grew linearly with the number of AST nodes as did the size of the AST binary format. AST Nodes are used in place of number of attributes in our analysis as we have found this to be a better measure of a policy's complexity than the number of attributes referenced. However, further testing is needed to evaluate the impact of requesting large number of administrative and environment attributes from the Policy Authority's Attribute Database or the User Service sending a large number of object attributes.

5.6 Conclusions & Future Work

We have introduced the first complete architecture for HGABAC that supports the full model and policy language in a distributed environment. This work aids in solving the problem of attribute storage and sharing first indeified in Chapter 2 Section 2.4.7 by detailing the supporting protocols and services for sharing attributes in the context of HGABAC. Each architecture component is detailed and the sequence of requests upon each service is described. An attribute certificate specification and encoding is introduced for securely sharing and proving ownership

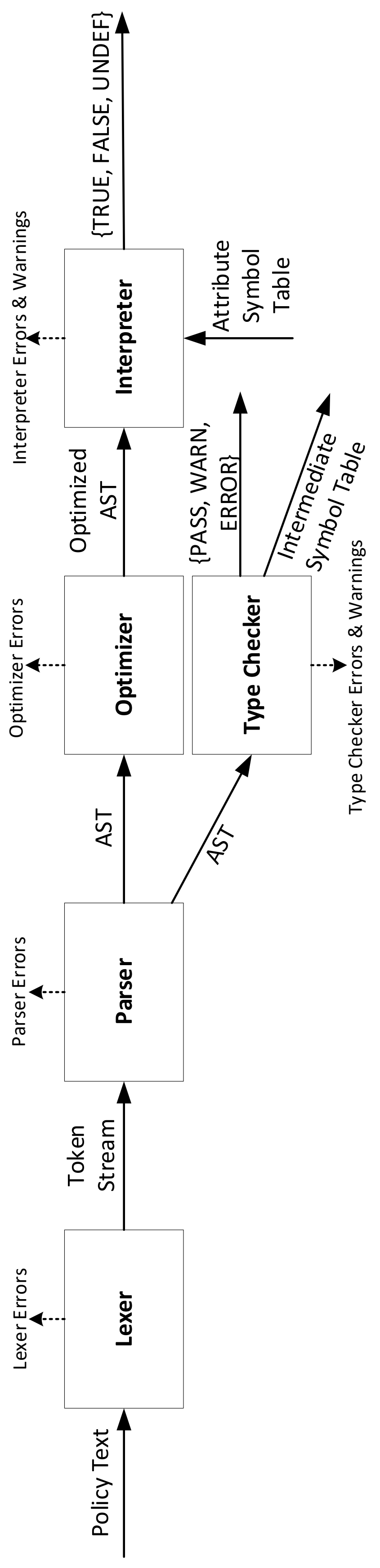


Figure 5.14: HGPLv2 interpreter modules and information flow.

of attributes. The HGPL policy language from Chapter 3 and [Servos and Osborn 2014] is updated to support policy references and our HGABAC Namespace for uniquely identifying attributes (as well as other HGABAC elements) across disparate security domains.

Details of a Python-based HGAA implementation are given and preliminary evaluation results are discussed. Each analysis done to date, shows a linear relationship with the number of attributes or number policy AST nodes, suggesting linear scalability. A number of possible optimizations are mentioned, including precomputing policy ASTs and intermediate symbol tables. Further evaluation of the architecture as a whole and under more diverse scenarios is needed in future work as well as comparisons to solutions utilizing generic architectures and standards (i.e. XACML, SAML, etc.) but preliminary results are promising.

Other directions for future work include extending the AC specifcation and architecture to support the delegation strategies discussed in Chapter 4. Work towards acomplishing this task is pursued in Chapter 6, in which HGABAC and HGAA are extened to support a user-to-user delegation model based on one of the strategies from Chapter 4. Creating an administrative model for HGABAC (or possibly utilizing the GURA$_G$ model presented in [Gupta and Sandhu 2016]) would allow for the creation of administrative services specified in HGAA but currently left unimplemented.

Currently, HGAA does not specify how attributes are stored or how the HGABAC group graph is represented in the Attribute Authority or Attribute Databases. While the described implementation uses a traditional SQL-based relational database, other approaches may be more advantageous for certain use cases. The Resource Description Framework (RDF)[Lassila et al. 1998] standard may provide an ideal means of expressing these attributes as RDF is built around *subject–predicate–object* statements known as semantic triples which may match well with the *user-attribute-value* relations that make-up attribute assignments in HGABAC. The use of RDF for storing and representing subject and object attributes for access control purposes has been explored to a limited extent by Stermsek et al. who encode these attributes in RDF metadata[Stermsek et al. 2004]. RDF and Resource Description Framework Schema (RDFS)[Brickley et al. 1999] may also provide an ideal means of representing the HGABAC group graph and sharing attributes (in place of or in combination with an HGAA Attribute Certificate) in part due to their ability to describe ontologies and hierarchical connections and incorporating them would be a beneficial direction for future work.

Bibliography

Anne Anderson, Anthony Nadalin, B Parducci, D Engovatov, H Lockhart, M Kudo, P Humenn, S Godik, S Anderson, S Crocker, et al. extensible access control markup language (XACML) version 1.0. Technical report, OASIS, 2003.

Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. Uniform resource identifier (URI): Generic syntax. STD 66, RFC Editor, January 2005. URL http://www.rfc-editor.org/rfc/rfc3986.txt.

Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. ABAC with group attributes and attribute hierarchies utilizing the policy machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 17–28. ACM, 2017.

Dan Brickley, Ramanathan V Guha, and Andrew Layman. Resource description framework (RDF) schema specification. 1999.

David Chadwick, Alexander Otenko, and Edward Ball. Role-based access control with x. 509 attribute certificates. *IEEE Internet Computing*, 7(2):62–69, 2003.

Dave Crocker and Paul Overell. Augmented BNF for syntax specifications: ABNF. Technical report, RFC 2234, November, 1997.

S. Farrell, R. Housley, and S. Turner. An internet attribute certificate profile for authorization. RFC 5755, RFC Editor, January 2010. URL http://www.rfc-editor.org/rfc/rfc5755.txt.

David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The policy machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

David Ferraiolo, Serban Gavrila, and Wayne Jansen. *Policy Machine: Features, Architecture, and Specification*. US Department of Commerce, National Institute of Standards and Technology, 2014.

Maanak Gupta and Ravi Sandhu. The GURA_G administrative model for user and group attribute assignment. In *International Conference on Network and System Security*, pages 318–332. Springer, 2016.

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), 2013.

John Hughes and Eve Maler. Security assertion markup language (SAML) v2.0 technical overview. Technical report, OASIS, 2005. SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08.

Xin Jin, Ram Krishnan, and Ravi S Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

Ora Lassila, Ralph R Swick, et al. Resource description framework (RDF) model and syntax specification. 1998.

ITUT Recommendation. ITU-T recommendation X.680 abstract syntax notation one (ASN.1): Specification of basic notation, 2015.

E Rissanen, H Lockhart, and T Moses. XACML v3.0 administration and delegation profile version 1.0. Technical report, 2009. Committee Draft.

Carlos E Rubio-Medrano, Clinton D'Souza, and Gail-Joon Ahn. Supporting secure collaborations with attribute-based access control. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*, pages 525–530. IEEE, 2013.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *International Symposium on Foundations and Practice of Security*, pages 187–204. Springer, 2014.

Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *International Symposium on Foundations and Practice of Security*, pages 320–328. Springer, 2016.

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017.

Daniel Servos and Sylvia L Osborn. HGAA: An architecture to support hierarchical group and attribute-based access control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control (ABAC'18)*, pages 1–12, March 2018.

Jakob Simon-Gaarde. *Ladon Webservice Framework*. URL http://ladonize.org. Accessed: 2017-12-15.

SQLAlchemy. *SQLAlchemy - The Database Toolkit for Python*. URL https://www.sqlalchemy.org/. Accessed: 2017-12-15.

Gerald Stermsek, Mark Strembeck, and Gustaf Neumann. Using subject-and object-specific attributes for access control in web-based knowledge management systems. *Proc. SKM*, pages 15–20, 2004.

J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. de Bruijn, C. de Laat, M. Holdrege, and D. Spence. AAA authorization framework. RFC 2904, RFC Editor, August 2000. URL http://www.rfc-editor.org/rfc/rfc2904.txt.

Lingyu Wang, Duminda Wijesekera, and Sushil Jajodia. A logic-based framework for attribute based access control. In *Proceedings of the 2004 ACM Workshop on Formal Methods in Security Engineering*, pages 45–55. ACM, 2004.

Chapter 6

Incorporating Off-Line Attribute Delegation into Hierarchical Group and Attribute-Based Access Control

Reprint Notice

A version of this chapter was published as part of the *The 12th International Symposium on Foundations & Practice of Security (FPS'2019)* as:

Daniel Servos and Michael Bauer. Incorporating off-line attribute delegation into hierarchical group and attribute-based access control. In *International Symposium on Foundations and Practice of Security*, 2019. Forthcoming publication in Springer's Lecture Notes in Computer Science LNCS

It is also a forthcoming publication in the Springer's Lecture Notes in Computer Science series *(volume number 12056)*. A preprint copy can be found at `http://cs1.ca/papers/attdel.pdf`

6.1 Introduction

Attribute-Based Access Control (ABAC) is an access control model in which users are granted access rights based on the attributes of users, objects, and the environment rather than a user's identity or predetermined roles. The increased flexibility offered by such attribute-based policies combined with the identityless nature of ABAC have made it an ideal candidate for the next generation of access control models. The beginnings of ABAC in academic literature date back as early as 2004 with Wang, et al's Logic-Based Framework for Attribute Based Access Control[Wang et al. 2004] and even earlier in industry with the creation of the eXtensible Access Control Markup Language (XACML)[Anderson et al. 2003] in 2003. However, it is only in recent years that ABAC has seen significant attention[Servos and Osborn 2017]. This renewed interest has lead to the development of dozens of ABAC models, frameworks and implementations, but to date, few works have touched on the subject of delegation or how it might be supported in attribute-based models.

Delegation is a key component of comprehensive access control models and has been indetified as an open problem for ABAC research (as detailed in Chapter 2 Section 2.4.6). Delegation enables users to temporarily and dynamically delegate their access control rights to another entity after policies have been set in place by an administrator. This ability allows users to adapt to the realities of everyday circumstances that are not possible to foresee during policy creation and is critical in domains such as healthcare[Rostad and Edsberg 2006]. ABAC brings new problems and complications when incorporating delegation that are not present in the traditional models (RBAC, DAC, MAC, etc.)[Servos and Osborn 2016] (as discussed in Chapter 4). In the traditional models, delegation is relatively straightforward, a set of permissions or role memberships (as in RBAC) is delegated directly by a delegator to a delegatee under set conditions (e.g. an expiry date or *"depth"* of delegation). In ABAC, this is complicated by identityless nature of ABAC (i.e. access control decisions being made on the basis of attributes rather then the user's identity) and the flexibility of attribute-based policies that may include dynamic attributes such as the current time, physical location of the user, or other attributes of the system's environment.

In a previous work[Servos and Osborn 2016] (Chapter 4), we offered a preliminarily investigation into the possible strategies for incorporating delegation into ABAC and the benefits and drawbacks of each method. A number of these proposed strategies have been further developed into working models by others, such as the work by Sabathein, et al.[Sabahein et al. 2018] towards creating a model of delegation for the Hierarchical Group and Attribute-Based Access Control (HGABAC)[Servos and Osborn 2014] (Chapter 3) model using our User-to-Attribute Group Membership Delegation Strategy[Servos and Osborn 2016] (Chapter 4 Section 4.2.2.2). In this chapter, we seek to explore and put forth a novel attribute-based delegation model based on an unutilized delegation strategy, User-to-User Attribute Delegation[Servos and Osborn 2016] (Chapter 4 Section 4.2.2.1). We offer both an extension to the HGABAC model to provide a theoretical blueprint for incorporating delegation as well as an extension to Hierarchical Group Attribute Architecture (HGAA)[Servos and Osborn 2018] (Chapter 5) to provide a practical means of implementing it. Unlike current efforts, a particular emphasis is placed on maintaining the identityless nature of ABAC as well as the ability to delegate attributes in an *"off-line"* manner (i.e. without the user having to connect to a third party to perform delegation).

The remainder of this chapter is organized as follows; Section 6.2 introduces the potential delegation strategies developed in our previous work and gives background on the HGAA and HGABAC model. Section 6.3 details our model of User-to-User Attribute Delegation and how it is incorporated into the HGABAC model. Section 6.4 provides a framework for supporting our delegation model and details modifications to the HGAA architecture to support it, including new extensions to the Attribute Certificate format to include delegation concepts. Finally, Section 6.5 gives concluding remarks and directions for future work.

6.2 Background

6.2.1 The HGABAC Model

HGABAC[Servos and Osborn 2014] (Chapter 3) offered a novel model of ABAC that introduced the concept of hierarchical user and object groups. Attributes are assigned both directly

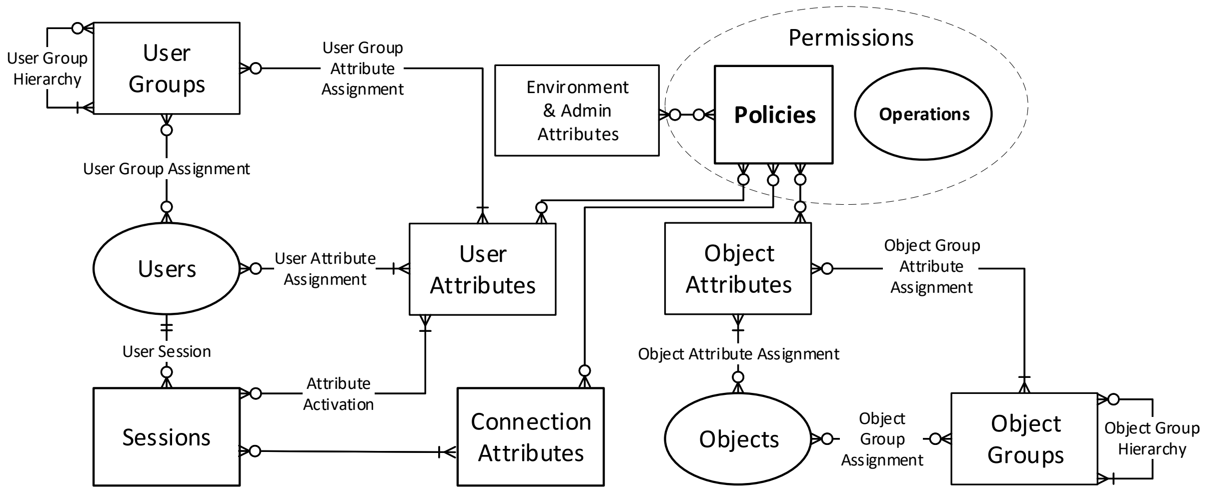


Figure 6.1: HGABAC model relations and components denoted in Crow's Foot Notation. Primitive components are shown in ovals.

to access control entities (e.g. users and objects) and indirectly assigned through groups. Users then have a *direct* set of attributes, directly assigned by an administrator, as well as an *inherited* set of attributes, indirectly assigned to them via their membership in one or more user groups. The set of attributes used for policy evaluations is the user's *effective* attribute set, that is, the set that is the result of merging their *direct* and *inherited* attribute sets. This style of group membership and attribute inheritance is also used to assign attributes indirectly to objects via objects membership in object groups. These relations and the basic elements of HGABAC are shown in Figure 6.1 and a brief description of the basic HGABAC entities, relations and functions are as follows:

- **Attributes:** attributes are defined as attribute name, type pairs, that is $att = (name, type)$, where *name* is a unique name for the attribute and *type* is a data type (e.g. string, integer, boolean, etc.). When assigned to entities via direct assignment (e.g. User Attribute Assignment) or groups (e.g. User Group Attribute Assignment) they are associated with a set of values being assigned for that attribute.

- **User, Object and Connection Attributes:** attributes are defined as attribute name, type pairs, that is $att = (name, type)$, where *name* is a unique name for the attribute and *type* is a data type (e.g. string, integer, boolean, etc.). When assigned to entities via direct assignment (e.g. User Attribute Assignment) or groups (e.g. User Group Attribute Assignment) they are associated with a set of values being assigned for that attribute.

- **Environment and Admin Attributes:** defined similarly to user and object attributes, but also have a set of values directly associated with them as they are not assigned to other entities. Defined as $att = (name, value, type)$ where *value* is a set of values conforming to the given data *type*.

- **Users (U):** entities that may request access on system resources through sessions.

- **Objects (O):** resources (files, devices, etc.) for which access may be limited.

```
P1:  /user/age >= 18 AND /object/title = "Adult Book"
P2:  /user/id = /object/author
P3:  /policy/P1 OR /policy/P2
P4:  /user/role IN "doctor", "intern", "staff" AND /user/id != /object/patient
```

P1 describes a policy requiring the user to be 18 or older and the title of the object to be “Adult Book”. P2 requires the user to be the author of the document. P3 combines policies P1 and P2, such that the policy is satisfied if either policy (P1 or P2) is satisfied. Finally, P4, requires a user’s role to be one of “doctor”, “intern”, or “staff” and that they are not listed as the patient.

Figure 6.2: Example HGABAC HGPLv2[Servos and Osborn 2018] (Chapter 5 Section 5.4) policies.

- **Operations (Op):** operations that may be applied to an object (read, write, etc.).
- **Policies (P):** policy strings following the Hierarchical Group Policy Language (HGPL).
- **Groups (G):** a hierarchical collection of users or objects to which attributes may be assigned. Group members inherit all attributes assigned to the group and the groups parents in the hierarchy. Defined as $g = (name, m \subseteq M, p \subseteq G)$ where *name* is the name of the group, m is the set of members, M is either the set of all Users or all Objects, p is the groups parents and G is the set of all groups.
- **Sessions:** sessions allow users to activate a subset of their effective attributes. This subset is used as the user attributes for policy evaluations. Sessions are represented as a tuple s in the form $s = (u \in U, a \subseteq effective(u \in U), con\_atts)$ where u is the user who owns the session, con_atts is the set of connection attributes for the session and a is the set of attributes the user is activating.
- **Permissions:** a pairing of a policy string and an operation in the form $perm = (p \in P, op \in Op)$. A user may perform an operation, op, on an object if there exists a permission that contains a policy, p, that is satisfied by the set of attributes in the user’s session, the object being accessed and the current state of the environment.
- **direct(x):** mapping of a group or user, x, to the attribute set they were assigned directly.
- **inherited(x):** mapping of a group or user, x, to the set of all attributes inherited from their group memberships and the group hierarchy.
- **effective(x):** mapping of a group or user, x, to the attribute set resulting from merging the entities directly assigned and inherited attribute sets.

The largest advantage of attribute groups is simplifying administration of ABAC systems, allowing administrators to create user or object groups whose membership indirectly assigns sets of attribute/value pairs to its members. The hierarchical nature of the groups, in which child groups inherit all attributes from their parent groups, allow for more flexible policies, administration and even emulation of traditional models such as RBAC, MAC and DAC when combined with the HGPL policy language.

Permissions in HGABAC take the form of operation/policy pairs, in which an operation is allowed to be performed if the policy is satisfied by the requesting user's active attribute set, attributes of the object being affected, the current state of the environment and a number of other attribute sources. Policies are defined in the Hierarchical Group Policy Language (HGPL), a C style language using ternary logic to define statements that result in *TRUE*, *FALSE* or *UNDEF*. Example HGPLv2 policies are given in Figure 6.2 and the full language is defined in [Servos and Osborn 2018] (Chapter 5 Section 5.4). These policies would be combined with an operation to form a permission. For example the permission $Perm_1 = (P1, read)$ would allow any user who is at least 18 years of age to read the object titled *"Adult Book"* based on the permission $P1$ from Figure 6.2. Similarly, the permission $Perm_2 = (P2, write)$ would allow any author of an object to write to that object.

6.2.2 Hierarchical Group Attribute Architecture (HGAA)

While HGABAC provides an underlying model for ABAC, a supporting architecture is still required to provide a complete system and facilitate use in real-world distributed environments. HGAA[Servos and Osborn 2018] (Chapter 5) provides a system architecture and implementation details for HGABAC that answer questions such as *"who assigns the attributes?"*, *"how are attributes shared with each party?"*, *"how does the user provide proof of attribute ownership?"*. and *"where and how are policies evaluated?"* that are often left unanswered by ABAC models alone. HGAA accomplishes this by adding five key components:

- **Attribute Authority (AA):** A service responsible for managing, storing and providing user attributes by issuing attribute certificates.
- **Attribute Certificate (AC):** A cryptographically secured certificate listing a user's active attributes for an HGABAC session as well as revocation information.
- **HGABAC Namespace:** A URI-based namespace for uniquely identifying attributes and HGABAC elements across multiple federated domains and authorities.
- **Policy Authority:** A service which manages and evaluates HGABAC policies on behalf of a user service provider.
- **User Service Provider:** A provider of a service to end users that has restricted access on the basis of one or more HGABAC policies (i.e. the service on which access control is being provided).

These components and the information flow between them are shown in Figure 6.3.

In a system following HGAA, users request Attribute Certificate (AC) from their home domain's Attribute Authority (AA) containing a list of their attributes for a session. Users may then use this AC to make requests on protected user services (both in their home domain or run by external organizations). These protected services verify the user's AC and check that the user has permission to access the service using their local domains Policy Authority. A key feature of the architecture is the separation of the AA from the other services. Once users are issued an AC, there is no longer a need for the user or other parties to contact the AA for the

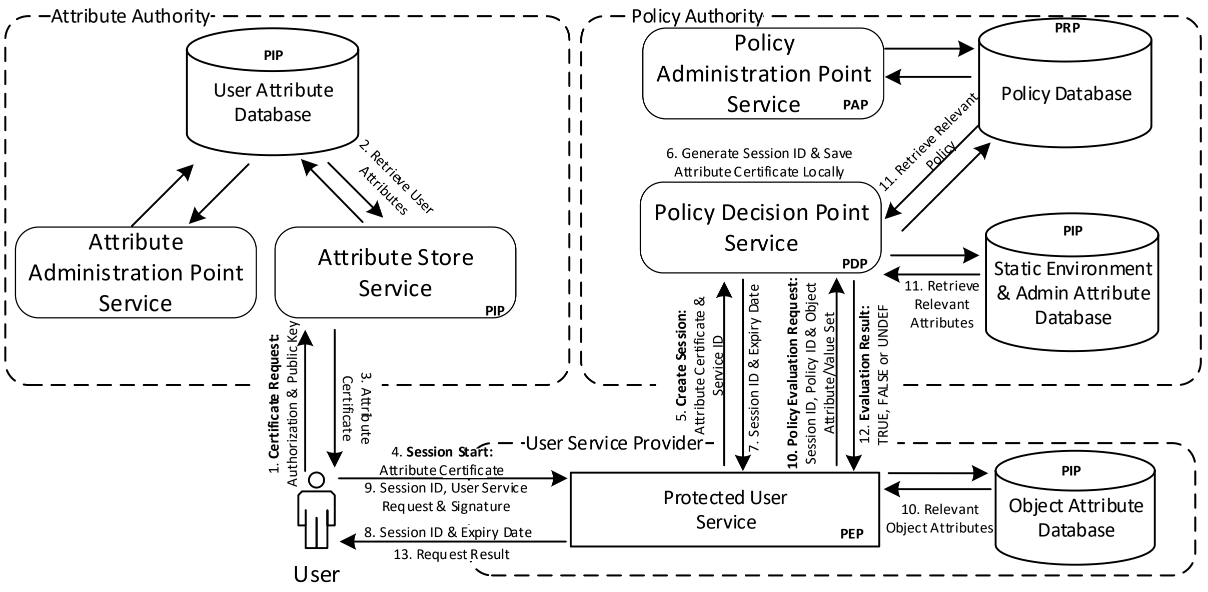

Attribute Authority
PIP
User Attribute Database
2. Retrieve User Attributes
Attribute Administration Point Service
Attribute Store Service
PIP
1. Certificate Request: Authorization & Public Key
3. Attribute Certificate
Policy Authority
Policy Administration Point Service
PAP
PRP
Policy Database
11. Retrieve Relevant Policy
6. Generate Session ID & Save Attribute Certificate Locally
Policy Decision Point Service
PDP
PIP
Static Environment & Admin Attribute Database
11. Retrieve Relevant Attributes
5. Create Session: Attribute Certificate & Service ID
7. Session ID & Expiry Date
10. Policy Evaluation Request: Session ID, Policy ID & Object Attribute/Value Set
12. Evaluation Result: TRUE, FALSE or UNDEF
User Service Provider
4. Session Start: Attribute Certificate
9. Session ID, User Service Request & Signature
8. Session ID & Expiry Date
13. Request Result
User
Protected User Service
PEP
10. Relevant Object Attributes
PIP
Object Attribute Database

Figure 6.3: HGAA services, components and information flow. Numbers indicate order of requests. Dotted lines denote components of a service (i.e. Attribute Store Service is a component/subservice of the Attribute Authority Service).

duration of the session. Separating services in this way simplifies the problem of user's using their attribute-based credentials across independent organizational boundaries. Services run by external organizations need not communicate with the user's home organization to verify their attributes beyond trusting the home organization's public key used to sign ACs.

The framework presented in Section 6.4 extends the AC format to add delegation related features. ACs are ideal for this purpose as room has been left for future extensions including space for delegation extensions that were not part of the original work. Section 6.4 also presents a modification to the HGAA protocol to add an optional delegation step in which users may delegate part of their certificate to another user.

6.2.3 Potential Delegation Strategies

In our previous work[Servos and Osborn 2016] (Chapter 4), we explored possible strategies for integrating delegation into ABAC and propose several potential methods primarily based on the access control element being delegated (e.g. attributes, group memberships, permissions, etc.). Each family of strategies results in unique proprieties and complications to overcome. The delegation model and architecture presented in the subsequent sections (Sections 6.3 and 6.4) of this chapter are based on the User-to-User Attribute Delegation strategy in which users acting as a delegator, delegate a subset of their user attributes to another user acting as the delegatee. The delegated attributes are merged with the delegatee's directly assigned attributes (i.e. assigned through any means but delegation) to form the delegatee's set of user attributes used in policy evaluations.

An example of this style of delegation is shown in Figure 6.4, in which Alice (the delegator) delegates a subset of their directly assigned attributes to the user Charlie (the delegatee) such

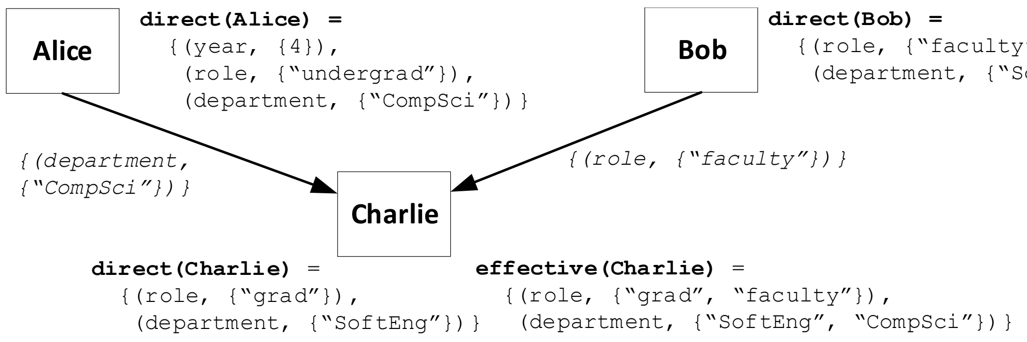

Alice
direct(Alice) = {(year, {4}), (role, {"undergrad"}), (department, {"CompSci"})}
Bob
direct(Bob) = {(role, {"faculty"}), (department, {"SoftEng"})}
{(department, {"CompSci"})}
{(role, {"faculty"})}
Charlie
direct(Charlie) = {(role, {"grad"}), (department, {"SoftEng"})}
effective(Charlie) = {(role, {"grad", "faculty"}), (department, {"SoftEng", "CompSci"})}

Figure 6.4: Example of User-to-User Attribute Delegation. Arrows denote direction of delegation (arrow points to delegatee), boxes represent users of the system.

that Charlie may satisfy the policy requiring the user to be in the Computer Science department (e.g. `user.department = "CompSci"`). At the same time, the user Bob (a second delegator) also delegates a subset of their attributes to Charlie such that they may satisfy a policy requiring the user to be a faculty member in the Software Engineering department (e.g. `user.role = "faculty" AND user.department = "SoftEng"`). In this example, both the attributes delegated by Alice (a *department* attribute with the value *"CompSci"*) and the attributes delegated by Bob (a *role* attribute with the value *"faculty"*) are combined with Charlie's directly assigned attributes to form their effective attribute set. Note that Bob only needed to delegate the *role* attribute as Charlie already had a *department* attribute with the value *"SoftEng"*.

While this style of attribute delegation may seem straightforward, our previous investigation[Servos and Osborn 2016] identified a number of potential issues regarding Attribute Delegation:

Conflicting Policy Evaluations: Merging attribute sets can lead to multiple values for an attribute. While this is an intended feature of HGABAC, it can lead to unintended conflicts when the values are a result of delegation as opposed to careful design. For example, the policy `user.department` ≠ `"SoftEng"` results in two different results for Charlie in Figure 6.4 depending on the value of *department* used.

User Collusion: Merging attribute sets allows users to combine their attributes such that they may satisfy policies they could not individually. In Figure 6.4, Alice and Charlie may collude to satisfy the policy `user.department = "CompSci" and user.role = "grad"`, if Alice delegates her *department* attribute such that Charlie satisfies the policy.

Loss of Attribute Meaning: A key ABAC feature is that attributes are descriptive of their subjects and policies are created with this in mind. Merging attribute sets leads to an effective set that is no longer descriptive of the user. In Figure 6.4, it is not clear what Charlie's role or department is based solely on their effective attribute set. While this makes delegation possible, it increases the difficulty of policy creation as all allowable delegations and combinations of attributes would need to be taken into account.

User Comprehension: User comprehension of policies, the attributes needed to satisfy them and the permissions they grant is a complex and open problem in ABAC[Servos and Osborn 2017] (Chapter 2). Delegation further complicates the issue, obscuring what attributes need

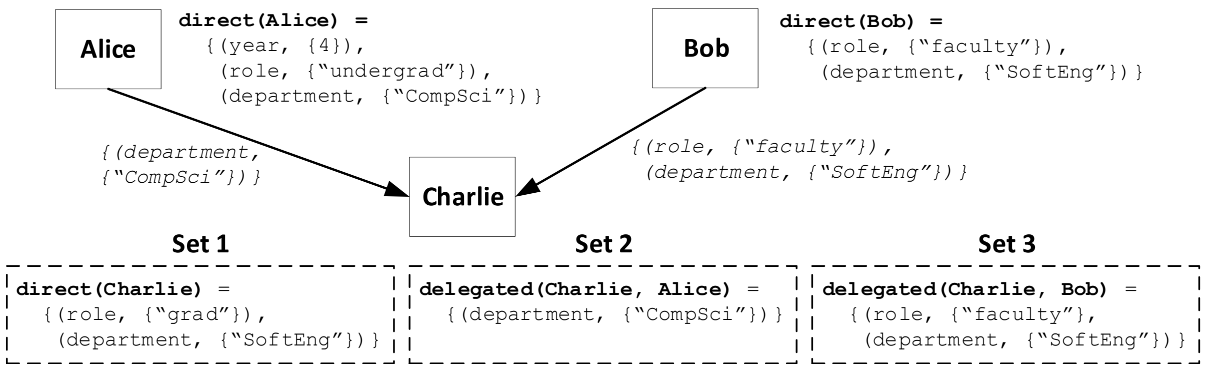

Alice
direct(Alice) = {(year, {4}), (role, {"undergrad"}), (department, {"CompSci"})}
Bob
direct(Bob) = {(role, {"faculty"}), (department, {"SoftEng"})}
{(department, {"CompSci"})}
{(role, {"faculty"}), (department, {"SoftEng"})}
Charlie
Set 1
direct(Charlie) = {(role, {"grad"}), (department, {"SoftEng"})}
Set 2
delegated(Charlie, Alice) = {(department, {"CompSci"})}
Set 3
delegated(Charlie, Bob) = {(role, {"faculty"}, (department, {"SoftEng"})}

Figure 6.5: Example of Isolated Attribute Delegation. Arrows denote delegation direction and solid boxes users. Charlie may activate one set shown in dashed boxes at a time.

to be delegated and the permission granted. When attributes sets are merged, users must also consider possible conflicts that could lead to granting unintended permissions.

To resolve these issues, the delegation model and architecture described in this chapter takes a modified approach to User-To-User Attribute Delegation in which the attributes delegated to a delegatee from delegators are isolated and not merged with the delegatee's directly assigned attribute set. A delegatee may then choose to activate either their own directly assigned attributes or their delegated attributes in a given session but never both at the same time. An example of this approach is shown in Figure 6.5. In this example, Alice still wishes to delegate their attributes such that Charlie may satisfy the policy `user.department = "CompSci"` and Bob still wishes Charlie to satisfy the policy `user.role = "faculty" AND user.department = "SoftEng"`. In Alice's case, they still only delegate their *department* attribute, however, now Charlie must choose between activating the directly assigned attribute set, *Set 1*, or the set delegated by Alice, *Set 2*. Charlie is unable to combine their own directly assigned attributes with those delegated by Alice and must activate the delegated attributes *(Set 2)* to satisfy the policy `user.department = "CompSci"`. Note that in this case Alice was only required to delegate a subset of their attributes to satisfy this policy.

In the case of Bob, it is now required that both Bob's *role* and *department* attributes are delegated to Charlie. In the previous example (Figure 6.4), Bob only needed to delegate their *role* attribute as Charlie was already assigned a *department* attribute with the value *"SoftEng"* and it was merged with the attributes delegated by Bob. With the attributes sets isolated, both attributes are required as Charlie may not merge them with his own. The policy `user.role = "faculty" AND user.department = "SoftEng"` will only be satisfied when the set delegated by Bob *(Set 3)* is activated.

Isolation of delegated attributes avoids conflicting policy evaluations (at least those caused by delegation) and user collusion as attributes sets are not merged. Attribute meaning is maintained to the extent that the active attribute set will be descriptive of either the delegatee or delegator. Issues with user comprehension, while still an open ABAC problem[Servos and Osborn 2017] (Chapter 2), are abated as the delegator can be ensured that regardless of the attributes delegated, the delegatee will not be able to satisfy any extra policy that they themselves are not able to. It is important to note that negative policies such as `role` $\neq$ `"undergrad"` are still problematic as a user could simply not delegate an attribute with a restricted value. This

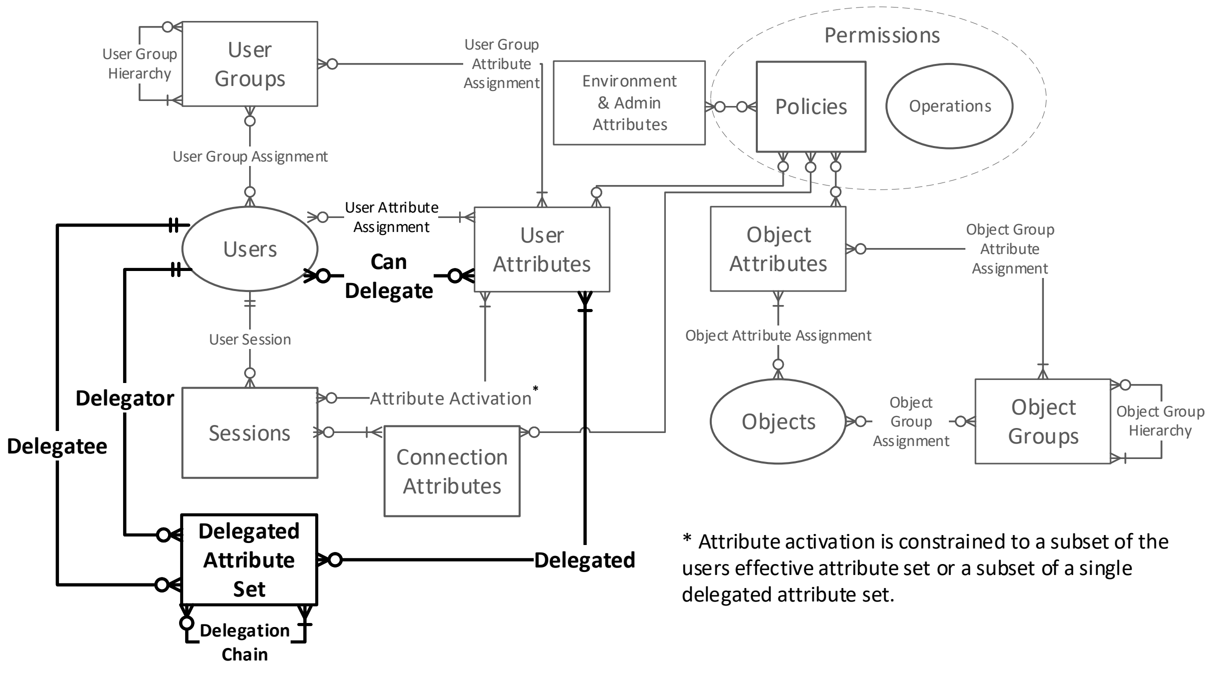


Figure 6.6: User-to-User Attribute Delegation extension to the HGABAC model. Added components are bold and black, original components are greyed out.

is however a problem with negative policies in all ABAC models that allow users to activate a subset of their attributes (such as HGABAC) and not simply one limited to attribute delegation.

6.3 Delegation Model

6.3.1 Delegated Attribute Set

Several extensions to HGABAC model are required to support User-to-User attribute style delegation. The most critical is the addition of the *Delegated Attribute Set* component (shown in Figure 6.6 with the other extensions), which contains the set of attributes delegated to a user in addition to the rules under which the delegation is permitted. This component is defined as follows where DAS is the set of all *Delegated Attribute Set*s in the system:

$$\begin{aligned}&\forall das \in DAS : \\ &\quad das = (delegatee \in U, delegator \in U, att\_set, depth \in \mathbb{N}_{\leq 0}, rule\_set \subseteq P, parent \in DAS)\end{aligned} \tag{6.1}$$

where *delegatee* is the unique ID of the user who is the recipient of the delegation, and *delegator* is the unique ID of the user who initiated the delegation. *att_set* is the set of attributes being delegated and their corresponding values (defined the same as attribute sets in the HGABAC model). The *att_set* is constrained to only containing attributes listed in *delegatable(delegator)* function (as defined in Section 6.3.2). *depth* is a positive integer selected by the delegator that describes how many more levels of delegation are permitted (e.g. if the user can further delegate these attributes on to another user). *rule_set* is the set of policies in the HGPL policy language selected by the delegator that must all evaluate to `TRUE` for the delegation to be maintained. Finally, *parent* is a reference to another *Delegated Attribute Set* in the case of subsequent delegations (see Section 6.3.3) or ∅ if this is the root of the delegation chain.

For example, the following *Delegated Attribute Set* would be created by the user Bob to delegate their role and department attribute to Charlie as shown in Figure 6.5.

$$\begin{aligned}
del\_att\_set = (\; & \\
& Charlie, \\
& Bob, \\
& \{ \\
& \quad (role, \{\text{“}faculty\text{”}\}), \\
& \quad (department, \{\text{“}SoftEng\text{”}\}) \\
& \}, \\
& 0, \\
& \{ \\
& \quad \text{“}/environment/date < 2020-04-12\text{”}, \\
& \quad \text{“}/connection/ip = 129.100.16.66\text{”} \\
& \}, \\
& \emptyset \\
)\; &
\end{aligned} \tag{6.2}$$

In this case, the delegation is constrained with two policies; `/environment/date < 2020-04-12` revokes the delegation if the current date is past April 12th, 2020 and `/connection/ip = 129.100.16.66` only makes this delegation valid if the delegatee is connecting from the IP address 129.100.18.66. A depth value of 0 limits the delegatee from further delegating these attributes onto other users. A null parent (∅) indicates that this is the first level of delegation and that Bob is first delegator in the chain.

6.3.2 Constraints on Delegatable Attributes

The set of attributes that may be assigned via delegation is not unlimited. There are two major constraints placed on the attributes and values a delegator may pass on to a delegatee. The first constraint is that delegator must have the delegated attribute and corresponding values in their effective attribute set (i.e. the set of attributes directly assigned to the delegator combined with those the delegator inherited from group membership). The second constraint placed on delegatable attributes comes from the new *Can Delegate (CD)* relation added to the HGABAC model (as shown in Figure 6.6). The *CD* relation allows a system administrator to directly constrain the set of attributes a user may delegate to a finite list and is defined as follows:

$$\begin{aligned}
&\forall cd \in CD : \\
&\qquad cd = (delegator \in U, att\_name \subseteq \{name | (name, type) \in UA\}, max\_depth \in \mathbb{N}_{\leq 0})
\end{aligned} \tag{6.3}$$

where *delegator* is the unique ID of the user who is permitted to delegate the set of user attributes listed in *att_name*, a list of unique user attribute names from the set of all user attributes (*UA*). *max_depth* is a positive integer value that limits the value of *deph* used in the *Delegated*

Attribute Set such that $depth \leq max\_depth$. A delegator may not select a *depth* larger than *max_depth* when delegating an attribute in the set *att_name*. A *max_depth* of 0 would limit the attributes from being further delegated.

The attributes a delegator may delegate is defined by the *delegatable* function which combines both constraints and maps a user to the set of attributes they may delegate:

$$\begin{aligned} delegatable(u) = \{\, att\_name \mid\; & (att\_name,\ values) \in effective(u) \\ & \wedge\ att\_name \in att\_set \\ & \wedge\ (u,\ att\_set,\ depth) \in CD \,\} \end{aligned} \tag{6.4}$$

where u is the ID of the delegator and *effective(u)* is the delegator's effective attributes set as defined by HGABAC. The result is an attribute only being delegatable if it is both assigned to the delegator normally (through User *Attribute Assignment* or *User Group Attribute Assignment*) and explicitly permitted via the *Can Delegate* relation.

6.3.3 Subsequent Delegations & Delegation Chains

In addition to the attributes listed in *delegatable(u)*, users may also further delegate attribute sets they have been delegated so long as the maximum depth has not been reached. If a user, u, wishes to delegate a set of attributes they have been delegated, $das_{old} \in DAS$, they create a new *Delegated Attribute Set*, das_{new}, such that:

$$\begin{aligned} das_{new} = (\; & \\ & delegatee \in U, \\ & u, \\ & att\_set_{new} \subseteq das_{old}.att\_set, \\ & depth < das_{old}.depth, \\ & rule\_set_{new} \supseteq das_{old}.rule\_set, \\ & das_{old} \\ ) & \end{aligned} \tag{6.5}$$

That is das_{new} must contain the same or a subset of the attributes of das_{old}, must have a depth less than the depth listed in das_{old}, must have a rule set that is more restrictive than das_{old} (i.e. must contain the same rules plus optionally any additional rules) and must list das_{old} as the parent. These conditions ensure that subsequent delegations in a delegation chain are always more restrictive than their parents, the maximum depth is maintained and that attribute sets remain isolated.

An example delegation chain is shown in Figure 6.7. In this case, the user Bob is delegating his *role* attribute with the value *"faculty"* and *department* attribute with the value *"SoftEng"* to the user Charlie. This is the same delegation as discussed in 6.3.1 and Bob creates the same *DAS* as shown in Equation (6.2) but with a *depth* of at least 1. This *DAS* is referred to as das_C. The key difference is that Charlie now further delegates a subset of these attributes on to Dave and Erin. In the case of Dave, Charlie delegates just the *department* attribute and in the case of Erin, only the *role* attribute. To accomplish this, the following *DAS*s are created, das_D for

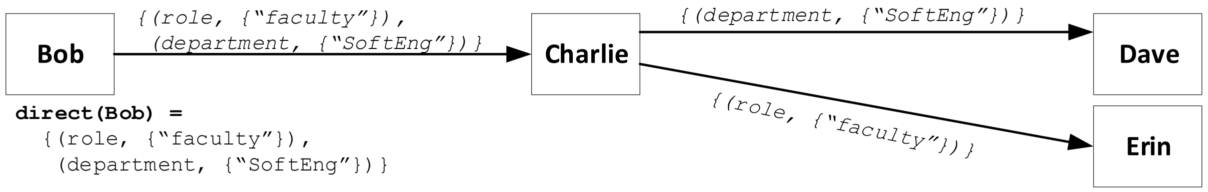


Figure 6.7: Example delegation chain. Bob delegates a *role* and *department* attribute to Charlie, who delegates the *department* attribute on to Dave and *role* attribute to Erin.

Dave and das_E for Erin:

$$\begin{aligned} das_D = (& \\ & Dave, \\ & Charlie, \\ & \{\ (department,\ \{\text{“}SoftEng\text{”}\})\ \}, \\ & 0, \\ & \{ \\ & \quad \text{“}/environment/date < 2020\text{–}04\text{–}12\text{”}, \\ & \quad \text{“}/connection/ip = 129.100.16.66\text{”}, \\ & \quad \text{“}/user/age >= 18\text{”} \\ & \}, \\ & das_C \\ ) & \end{aligned} \tag{6.6}$$

$$\begin{aligned} das_E = (& \\ & Erin, \\ & Charlie, \\ & \{\ (role,\ \{\text{“}Faculty\text{”}\})\ \}, \\ & 0, \\ & \{ \\ & \quad \text{“}/environment/date < 2020\text{–}04\text{–}12\text{”}, \\ & \quad \text{“}/connection/ip = 129.100.16.66\text{”}, \\ & \quad \text{“}/environment/date < 2020\text{–}04\text{–}01\text{”} \\ & \}, \\ & das_C \\ ) & \end{aligned} \tag{6.7}$$

das_D delegates the *department* attribute to Dave, but also adds in a new constraint on the delegation, `/user/age >= 18`, which requires that the user of this attribute set must have an *age* attribute in their effective attribute set (*effective(u)*) with a value equal to or greater than 18 for this delegation to be valid. All other constrains from das_C are present in das_D as required by Equation (6.5). If the *depth* in das_C was greater than 0, and Dave further delegated this attribute set on to another user, the `/user/age >= 18` constraint would have to be maintained, requiring all future delegatees in the chain to also be 18 years or older to use the delegated attributes.

The das_E set delegates the *role* attribute to Erin, but also adds an additional constraint of `/environment/date < 2020-04-01` which invalidates the delegation after April 1st, 2020. It is important to note that this does **not** conflict with the existing rule, `/environment/date < 2020-04-12`, from the parent set, das_C, but further constrains it as all policy rules must evaluate to `TRUE` for the delegation to be valid. In this way, subsequent delegators may tighten constrains on delegations but not loosen them.

Cycles in the delegation chain are permitted but not useful as each child in the chain must have the same or stricter constraints. The impact of such cycles is negligible as delegated attributes are isolated from each other and the user's effective attribute set as only one such set may be activated in a given session as discussed in Section 6.3.4. Cycles are prevented from being infinite in length as the *depth* of each set in the chain must be less than that of the parent and eventually reach 0, preventing further delegation.

6.3.4 Sessions & Attribute Activation

An important feature of the proposed User-to-User Attribute Delegation model is the isolation of delegated attributes from the user's effective set of attributes as well from other delegated attribute sets. This is accomplished through a modification of HGABAC's session definition. In the original HGABAC model, sessions are defined as a tuple of the form $s = (u \in U,\ att\_set \subseteq effective(u),\ con\_atts)$ where u is the user the session belongs to, att_set is the subset of the user's effective attributes being activated for this session and con_atts is the set of connection attributes that describe this session (e.g. IP address, time the session was started, etc.). To support delegation, we update the definition of a session to the following:

$$\begin{aligned} s = (& \\ & u \in U, \\ & att\_set_{effective} \subseteq effective(u) \lor att\_set_{delegated} \in \{\, del\_att\_set \mid \\ & \qquad das \in DAS \land das = (u,\ delegator,\ del\_att\_set,\ depth,\ rule\_set,\ parent)\,\}, \\ & con\_atts \\ ) & \end{aligned} \tag{6.8}$$

Or more simply put, the activated attribute set in a session may now be one of $att\_set_{effective}$ or $att\_set_{delegated}$ where $att\_set_{effective}$ is any subset of the user's effective attribute set (as per the original HGABAC session definition) and $att\_set_{delegated}$ is one of the delegated attribute sets delegated to the user via the new *Delegated Attribute Set* component. This limits users to either using their normally assigned attributes or **one** of their delegated attributes sets at a time, eliminating or vastly reducing the issues discussed in Section 6.2.2 related to merging delegated attribute sets.

6.3.5 Revocation

An important feature of the HGAA architecture is maintaining a separation between the Attribute Authority (AA) which grant attributes to users (via Attribute Certificate (AC)), and the Policy Enforcement Points and Policy Decision Points. This separation provides an important advantage in distributed and federated systems as no communication is required between the

AAs and the services their attributes grant access to beyond a user passing on their AC. This, however, raises a number of issues when it comes to revocation. As direct communication between the AAs and other services is optional, an AA's (and by extension its user's) ability to revoke delegated attributes is limited to the predefined delegation rules in the *rule_set* component of the *DAS*. As is shown in the example *DAS*s (in Equations (6.2), (6.6) and (6.7)) these rules may be any valid HGPL policy. If all policies evaluate to *TRUE* the delegation is valid, if the result is *FALSE* or *UNDEF* it is considered to be revoked.

In cases of delegation chains (as shown in Figure 6.7), if any policy in a *rule_set* is invalidated all subsequent delegations in the chain are also revoked. This is in part a consequence of all *rule_set*s in subsequent delegations being required to be a superset of the parent *rule_set* as is stated in Equation (6.5) (i.e. they must contain at least all policies in the parent rule set), but it is further required that each user in the chain satisfies the policies in their own *rule_set*. For example, if the policy `/user/age >= 18` is made a condition of a delegation from Bob to Charlie and Charlie subsequently delegates the attributes on to Dave, both Charlie and Dave must have their own *age* attribute that has a value of 18 or greater. The value of environment and administrator attributes are determined based on their current value at the Policy Decision Point. As the value of connection attributes for parents in the delegation chain may be unknown or undefined, how they are evaluated is left as an implementation decision (i.e. conditions involving connection attributes of parent users can be assumed to be *TRUE*, *UNDEF*, based on the last known values, or based on their values at the time of delegation).

Formally, we define the recursive function *active* which takes a Delegatable Attribute Set, *das*, and returns *TRUE* if the delegation is active (not revoked) and *FALSE* if the delegation is considered to be revoked.

$$active(das) = \begin{cases} active(das_{parent}) & \text{if } das.parent \neq \emptyset \\ \quad \bigwedge das.depth < das.parent.depth & \\ \quad \bigwedge das.depth \geq 0 & \\ \quad \bigwedge das.att\_set \subseteq delegatable(das.delegator) & \\ \quad \bigwedge das.rule\_set \supseteq das.parent.rule\_set & \\ \quad \bigwedge \forall rule \in das.rule\_set : valid(rule,\ das.delegatee) = TRUE & \\ & \\ das_{att\_set} \subseteq delegatable(das.delegator) & \text{if } das.parent = \emptyset \\ \quad \bigwedge \forall rule \in das.rule\_set : valid(rule,\ das.delegatee) = TRUE & \end{cases} \tag{6.9}$$

where *valid* is a HGABAC function which takes an HGPL policy and a user and returns *TRUE* if the user satisfies that policy for the current value of that user's attributes (including connection attributes) and the current state of the system (environment attributes and administrator attributes), *FALSE* if the policy is violated and *UNDEF* if the policy cannot currently be evaluated.

A secondary means of revocation is possible through HGAA's optional AC revocation lists. In HGAA, each AA may publish a revocation list that includes the serial number of any revoked AC issued by the authority. Policy Decision Points may optionally request this list either on demand or periodically depending on the nature of their service and if communication with the AA is possible. In the delegation framework detailed in the next section (Section 6.4), DAS are represented as special Delegated Attribute Certificates (DAC). These DACs may be revoked by the same mechanism. If a revoked DAC is part of a delegation chain, all subsequent delegations are also revoked.

6.4 Delegation Framework

The proceeding section (Section 6.3) laid out the theoretical delegation model and extensions to HGABAC to incorporate User-To-User Attribute Delegation. This section seeks to provide more practical details for how this delegation model may be implemented by extending HGAA to create a supporting delegation framework. Two key aspects of HGAA need to be expanded; the Attribute Certificate (AC) format to include delegation extensions and rules (detailed in Section 6.4.2), and the communication steps between users and services to provide a full certificate chain (detailed in Section 6.4.1).

6.4.1 Protocol Additions

In HGAA, users are issued an AC from an AA's Attribute Store Service. This document provides proof of a user's attributes in a cryptographically signed document as well as providing a mechanism for single sign-on and authentication with remote services. The AC format includes a listing of the user's attributes, details of the issuing authority, a public key assigned to the user, a range of dates for which the certificate is valid and a number of areas reserved for future extensions. User's prove ownership of an AC via a private key corresponding to the public key embedded in the AC. As the certificate is signed and contains all information about the user to base policy decision on, direct communication between the service being accessed and the AA is not required.

In the original architecture, after being issued an AC, users use the certificate to make requests on services. To support delegation, an additional delegation step is needed (as shown in Figure 6.8 as step 4). Rather than directly querying services, a user may now delegate all or a subset of the attributes in their AC to a third party by issuing a new AC called a Delegated Attribute Certificates (DAC). The DAC is identical to an AC issued by an AA but lists the delegator as the issuer and signer (rather than an AA), and the delegatee as the holder. The extensions to the AC format to support DACs and delegation, detailed in Section 6.4.2, enable the delegator to include delegation rules to trigger revocation (as discussed in Section 6.3.5), set a maximum delegation depth for subsequent delegations and select what subset of the attributes in their AC will be contained in the DAC (and delegated to the delegatee).

To complete the delegation, the delegator sends their AC and the new DAC to the delegatee. The delegatee validates both by checking the following:

1. The original components of the AC are valid as described in [Servos and Osborn 2014] (Chapter 3) (i.e. correctly signed by the AA, that the AC has not expired, etc.)

2. The *ACHolder* from the AC is the *ACIssuer* in the DAC (same UID, key, etc.).

3. The *ACHolder* given in the DAC is the delegatee (correct UID, public key, etc.).

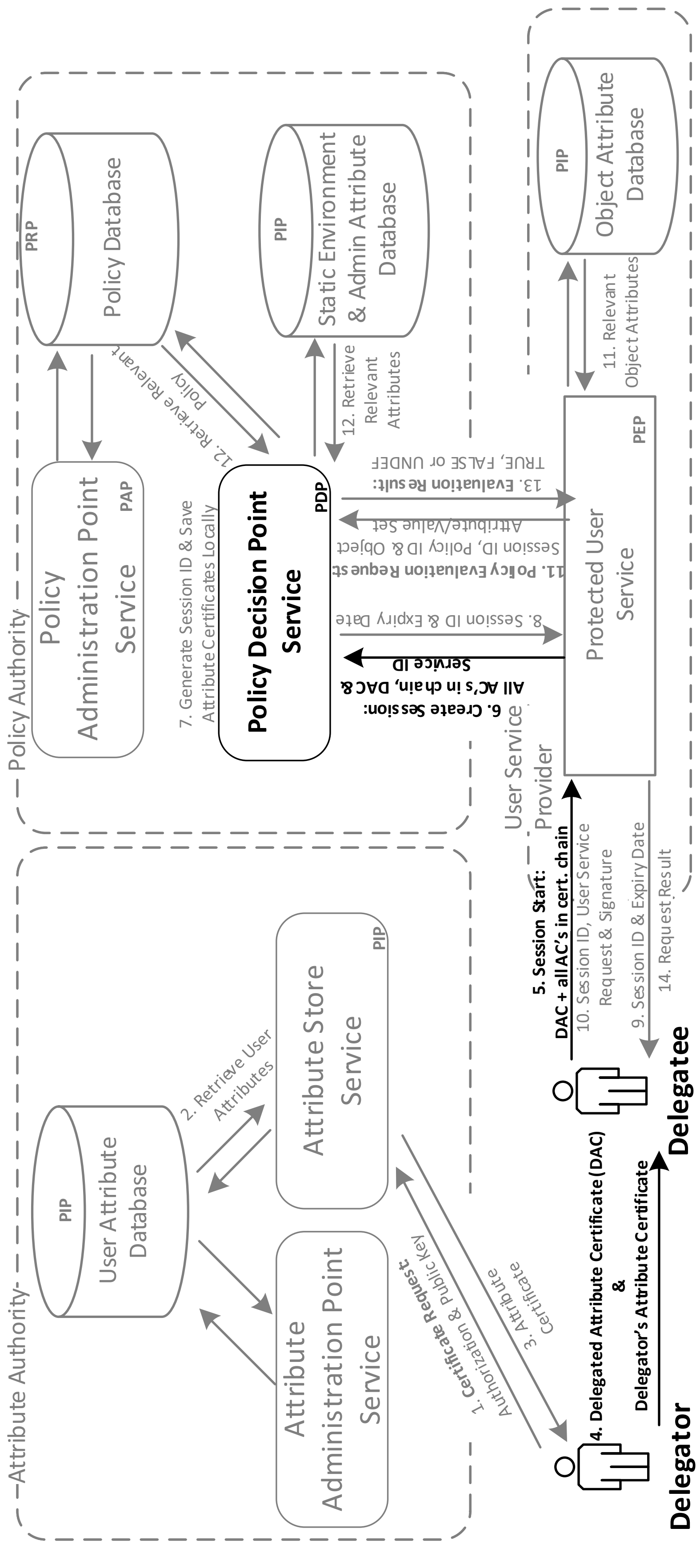

Attribute Authority
PIP
User Attribute Database
Attribute Administration Point Service
Attribute Store Service
PIP
2. Retrieve User Attributes
1. Certificate Request: Authorization & Public Key
3. Attribute Certificate
4. Delegated Attribute Certificate (DAC) & Delegator's Attribute Certificate
Delegator
Delegatee
Policy Authority
PRP
Policy Database
Policy Administration Point Service
PAP
PIP
Static Environment & Admin Attribute Database
12. Retrieve Relevant Policy
12. Retrieve Relevant Attributes
7. Generate Session ID & Save Attribute Certificates Locally
Policy Decision Point Service
PDP
6. Create Session: All AC's in chain, DAC & Service ID
8. Session ID & Expiry Date
11. Policy Evaluation Request: Session ID, Policy ID & Object Attribute/Value Set
13. Evaluation Result: TRUE, FALSE or UNDEF
User Service Provider
Protected User Service
PEP
PIP
Object Attribute Database
11. Relevant Object Attributes
5. Session Start: DAC + all AC's in cert. chain
10. Session ID, User Service Request & Signature
9. Session ID & Expiry Date
14. Request Result

Figure 6.8: Updated HGAA protocol to support delegation step. New and modified components shown in bold dark black, pre-existing HGAA components shown in light grey.

4. All attributes listed in the DAC are also found in the AC and have a *maxDepth* greater than zero in the AC.

5. All attributes in the DAC have the delegator listed as the delegator and a *maxDepth* less than or equal to the *maxDepth* in the DAC for that attribute.

6. The *ACRevocationRules* in the DAC are the same or stricter than in the AC.

7. The *ACDelegationRules* in the DAC are the same or stricter than in the AC.

8. The overall delegation *depth* in the DAC is less than the delegation *depth* in the AC and greater than or equal to 0.

9. That the delegation has not been revoked (i.e. all delegation rules return *TRUE*).

10. The DAC is signed by the delegator with the public key listed in the *ACHolder* sequence of the AC and the *ACIssuer* sequence of the DAC.

These checks enforce the rules on DASs described in Section 6.3.1 and ensure the delegation has not been revoked (as per Section 6.3.5). If the AC and DAC are valid, the delegatee may make requests upon services by sending both the AC and DAC with their request. The remainder of the HGAA protocol remains the same, but with the DAC being sent with the AC in steps 5 and 6 (Figure 6.8). The Policy Decision Point also makes the same checks (as listed above) on the DAC when validating the deletagee's attributes.

Subsequent delegations by the delegatee, to further delegatees, are supported. In such cases, the delegatee becomes the delegator and issues a new DAC using the processes previously described (their existing DAC becoming the AC and they become the issuer of the new DAC). This creates a chain of certificates leading back to the AA, each certificate being signed by the parent delegator. This process is shown in the Low Level Certificate Chain Diagram found in Figure 6.9. To allow services and the Policy Decision Point to verify subsequent delegations, each certificate in the chain is included with the first request upon a service and each certificate is validated.

6.4.2 Attribute Certificate Delegation Extensions

To incorporate our delegation model and updated HGAA protocol, several extensions to the AC format are required (described in Listing 6.1). The *Attribute* sequence is extended to include a *maxDepth* and *delegatorUniqueIdentifier* value for each attribute in the certificate. *delegatorUniqueIdentifier* states the ID of the original delegator (first in the chain) of the attribute or no value if not delegated. *maxDepth* corresponds to the *Can Delegate* relation (defined in Section 6.3.2) and has a value equal to 0 if this attribute cannot be delegated, 255 if there is no limit on the delegation depth or some value between 1 and 254 equal to the maximum depth allowed for this specific attribute.

Delegation rules from the DAS (defined in Section 6.3.1) are encoded in a new *DACDelegationRule* sequence which contains a HGPLv2 policy for each rule. The *depth* value from the DAS is included in a new instance of the *ACExtension* sequence in addition to a record of the original AA and the serial number of each certificate in the chain.

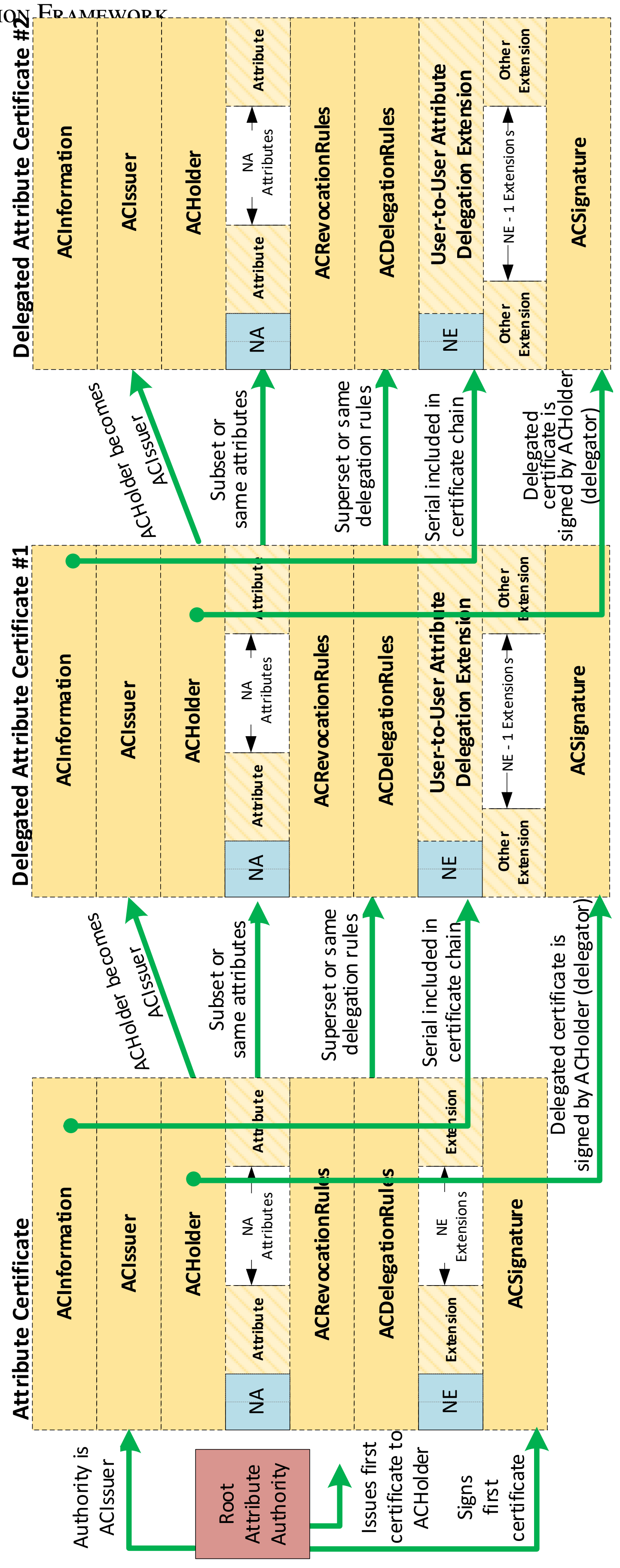

Root Attribute Authority
Authority is ACIssuer
Issues first certificate to ACHolder
Signs first certificate
Attribute Certificate
ACInformation
ACIssuer
ACHolder
NA
Attribute
NA Attributes
ACRevocationRules
ACDelegationRules
NE
Extension
NE Extensions
ACSignature
ACHolder becomes ACIssuer
Subset or same attributes
Superset or same delegation rules
Serial included in certificate chain
Delegated certificate is signed by ACHolder (delegator)
Delegated Attribute Certificate #1
User-to-User Attribute Delegation Extension
Other Extension
NE - 1 Extensions
Delegated Attribute Certificate #2

Figure 6.9: Low level Attribute Certificate chain diagram.

Listing 6.1: Updates to the AC format to support Attribute Delegation written in ASN.1 notation. Bold text indicate additions. Only updated sequences are shown.

```
Attribute ::= SEQUENCE {
        attributeID     OBJECT IDENTIFIER,
        attributeType   OBJECT IDENTIFIER,
        attributeValue  ANY DEFINED BY attributeType OPTIONAL,
        attributeName    VisibleString OPTIONAL,
        maxDepth INTEGER(0..255),
        delegatorUniqueIdentifier OBJECT IDENTIFIER OPTIONAL,
}

ACDelegationRules ::= SEQUENCE {
        SEQUENCE OF DACDelegationRule
}

DACDelegationRule ::= SEQUENCE {
        HGPLv2Policy VisibleString
}

– One instance of ACExtension with the following values
UToUAttDelv1 ACExtension ::= SEQUENCE {
        extensionID "ext:UToUAttDelv1",
        depth INTEGER(0..254),
        rootAuthorityUniqueIdentifier OBJECT IDENTIFIER,
        SEQUENCE OF DACCertificateSerial
}

DACCertificateSerial ::= SEQUENCE {
        certificateSerial INTEGER
}
```

The extended AC is kept backwards compatible with the original AC format by only updating sections marked for future extension. The changes have a minimal impact on the certificate size, adding at worst $3 + U$ bytes per attribute (where U is the size of the largest delegator ID), $2 * P$ bytes per delegation rule (where P is the maximum length of a HGPL policy), and $1+S$ bytes per certificate in the chain (where S is the serial number size in bytes). A byte level representation of the changes made to the AC is found in Figure 6.10.

6.5 Conclusions & Future Work

We have introduced the first model of User-to-User Attribute Delegation as well as a supporting architecture to aid implementation. Extensions to the HGABAC model (Section 6.3) add relations for authorizing what attributes can be delegated (*Can Delegate*) and to what depth. A new access control element, the *Delegated Attribute Set*, is added for representing current delegations in the system and the restrictions placed on them. Delegated attributes are kept isolated to prevent issues with Attribute Delegation, including user collusion and unexpected side effects on policy evaluations.

Updates to the HGAA protocol and AC format have been made (Section 6.4) to support the extended HGABAC model, including low level descriptions (Figures 6.9 and 6.10). These changes to the AC format are minimal in size, scaling with the number of attributes, delegation rules, and certificates in the chain. As changes have only been made to sequences marked for future expansion, the extended AC format remains backwards compatible with the original

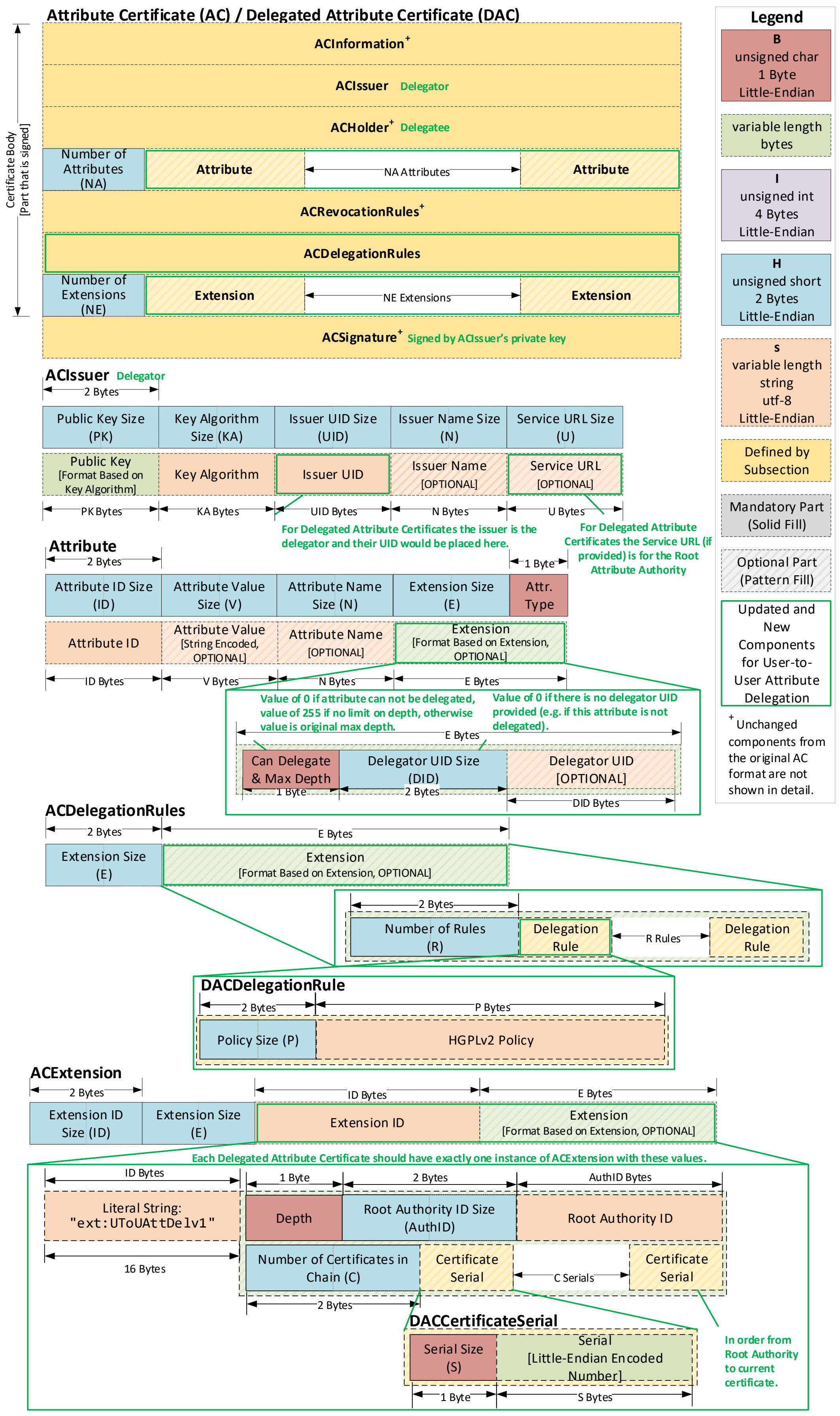

Attribute Certificate (AC) / Delegated Attribute Certificate (DAC)
Certificate Body [Part that is signed]
ACInformation+
ACIssuer Delegator
ACHolder+ Delegatee
Number of Attributes (NA)
Attribute
NA Attributes
Attribute
ACRevocationRules+
ACDelegationRules
Number of Extensions (NE)
Extension
NE Extensions
Extension
ACSignature+ Signed by ACIssuer's private key
Legend
B unsigned char 1 Byte Little-Endian
variable length bytes
I unsigned int 4 Bytes Little-Endian
H unsigned short 2 Bytes Little-Endian
s variable length string utf-8 Little-Endian
Defined by Subsection
Mandatory Part (Solid Fill)
Optional Part (Pattern Fill)
Updated and New Components for User-to-User Attribute Delegation
+ Unchanged components from the original AC format are not shown in detail.
ACIssuer Delegator
2 Bytes
Public Key Size (PK)
Key Algorithm Size (KA)
Issuer UID Size (UID)
Issuer Name Size (N)
Service URL Size (U)
Public Key [Format Based on Key Algorithm]
Key Algorithm
Issuer UID
Issuer Name [OPTIONAL]
Service URL [OPTIONAL]
PK Bytes
KA Bytes
UID Bytes
N Bytes
U Bytes
For Delegated Attribute Certificates the issuer is the delegator and their UID would be placed here.
For Delegated Attribute Certificates the Service URL (if provided) is for the Root Attribute Authority
Attribute
2 Bytes
1 Byte
Attribute ID Size (ID)
Attribute Value Size (V)
Attribute Name Size (N)
Extension Size (E)
Attr. Type
Attribute ID
Attribute Value [String Encoded, OPTIONAL]
Attribute Name [OPTIONAL]
Extension [Format Based on Extension, OPTIONAL]
ID Bytes
V Bytes
N Bytes
E Bytes
Value of 0 if attribute can not be delegated, value of 255 if no limit on depth, otherwise value is original max depth.
Value of 0 if there is no delegator UID provided (e.g. if this attribute is not delegated).
E Bytes
Can Delegate & Max Depth
Delegator UID Size (DID)
Delegator UID [OPTIONAL]
1 Byte
2 Bytes
DID Bytes
ACDelegationRules
2 Bytes
E Bytes
Extension Size (E)
Extension [Format Based on Extension, OPTIONAL]
2 Bytes
Number of Rules (R)
Delegation Rule
R Rules
Delegation Rule
DACDelegationRule
2 Bytes
P Bytes
Policy Size (P)
HGPLv2 Policy
ACExtension
2 Bytes
ID Bytes
E Bytes
Extension ID Size (ID)
Extension Size (E)
Extension ID
Extension [Format Based on Extension, OPTIONAL]
Each Delegated Attribute Certificate should have exactly one instance of ACExtension with these values.
ID Bytes
1 Byte
2 Bytes
AuthID Bytes
Literal String: "ext:UToUAttDelv1"
Depth
Root Authority ID Size (AuthID)
Root Authority ID
16 Bytes
Number of Certificates in Chain (C)
Certificate Serial
C Serials
Certificate Serial
2 Bytes
DACCertificateSerial
Serial Size (S)
Serial [Little-Endian Encoded Number]
1 Byte
S Bytes
In order from Root Authority to current certificate.

Figure 6.10: Byte level representation of updated Attribute Certificate. Only updated sections are shown.

HGAA AC. Care has been given to ensure that delegation is preformed in an *"off-line"* manner, without the need to contact a third party, to maintain the distributed nature of HGABAC and HGAA. However, support for *"off-line"* delegation comes at the cost of revocation flexibility and limits the possibility for real time revocation invoked by the delegator. To combat this, HGPL policies are used to embed delegation rules that trigger revocation as described in Section 6.3.5.

This work is part of an ongoing effort towards introducing delegation to ABAC and directions for future work will follow this path. To date, models for User-To-User Attribute Delegation (this chapter) and User-to-Attribute Group Membership Delegation [Sabahein et al. 2018] have been completed. The next steps will be to create models for the remaining strategies described in [Servos and Osborn 2016] (Chapter 4) in addition to reference implementations such that they can be fully explored, evaluated and compared. Directions for the User-To-User Attribute Delegation model include exploring the use of a *"Can Receive"* relation for users in place of the *"Can Delegate"* relation for attributes and experimenting with adding constraints that prevent specified users form being delegated a restricted attribute (e.g. to prevent certain users from stratifying a policy via delegation). Such *"Can Receive"* relations have been used successfully in RBAC delegation models[Crampton and Khambhammettu 2008] and have been shown to be more flexible. Work is needed to see if the same will hold true for delegation in ABAC. Finally, a more thorough evaluation of our delegation model is planned that will involve both formal validation (safety analysis) and experimental evaluation (reference implementation). Evaluating the overhead and impact of the size of delegation chains on certificate verification as well as the impact of the size and complexity of delegation rules will be an important future direction for such experimental evaluation. Delegatable Attribute-Based Anonymous Credentials (DAAC)[Blömer and Bobolz 2018] and related works that do not require the verification of certificate chains may provide an alternative to the HGAA Attribute Certificate. Determining if it is possible to incorporate DAAC into HGAA's *"off-line"* design will also be an important area to investigate in future efforts.

Bibliography

Anne Anderson, Anthony Nadalin, B Parducci, et al. eXtensible Access Control Markup Language (XACML) Version 1.0. *OASIS*, 2003.

Johannes Blömer and Jan Bobolz. Delegatable attribute-based anonymous credentials from dynamically malleable signatures. In *International Conference on Applied Cryptography and Network Security*, pages 221–239. Springer, 2018.

Jason Crampton and Hemanth Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2):123–136, 2008.

Lillian Rostad and Ole Edsberg. A study of access control requirements for healthcare systems based on audit trails from access logs. In *22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 175–186. IEEE, 2006.

Khaled Sabahein, Brian Reithel, and Feng Wang. Incorporating delegation into ABAC: Healthcare information system use case. In *Proceedings of the International Conference on Security and Management (SAM)*, pages 291–297, 2018.

Daniel Servos and Michael Bauer. Incorporating off-line attribute delegation into hierarchical group and attribute-based access control. In *International Symposium on Foundations and Practice of Security*, 2019. Forthcoming publication in Springer's Lecture Notes in Computer Science LNCS.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *International Symposium on Foundations and Practice of Security*, pages 187–204. Springer, 2014.

Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *International Symposium on Foundations and Practice of Security*, pages 320–328. Springer, 2016.

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017.

Daniel Servos and Sylvia L Osborn. HGAA: An architecture to support hierarchical group and attribute-based access control. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 1–12, 2018.

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

Chapter 7

Conclusions and Future Work

7.1 Reverse Literature Search

To show the significances and impact the research in this thesis has made to date, a brief *"reverse literature search"* is presented in this section of the publications that were influenced by or built on the work presented in Chapters 3 to 5 (Chapter 6 was not yet published at the time of writing but has since been presented at *The 12th International Symposium on Foundations & Practice of Security (FPS'2019)*).

7.1.1 Extensions to and Reformalizations of HGABAC

Since its publication in 2014, HGABAC has attracted the interest of a number of authors in the ABAC field. Several have put forth independent efforts to extend or otherwise reformalize the HGABAC model to add new capabilities or support addition access control concepts (e.g. administration). Of note is Gupta and Sandhu's work [Gupta and Sandhu 2016] towards creating an administrative model for HGABAC, entitled (GURA_G), built upon the preexisting Generalized URA Model (GURA) model [Jin et al. 2012a] for User-Role Assignment (URA). GURA_G is comprised of three sub-models, *UAA* (for the assignment of user attributes to users), *UGAA* (for the assignment of user attributes to groups), and *UGA* (for the assignment of users to groups). GURA_G uses special administrative roles (as do GURA and URA97 [Sandhu et al. 1999] upon which GURA_G is based) to assign administrative permissions to system administrators. These permissions grant the administrator the ability to add or delete attributes from the *UAA* or *UGAA* assignments and to remove or add users to the *UGA* assignment. Different administrative roles may have distinct conditions placed on these permissions. While the (GURA_G) model does not yet touch on object attributes or object attribute groups (although this is stated as a possible direction for future work), it is currently the best, and only, effort towards an HGABAC specific administrative model.

A second work of note is the undertaking by Bhatt et al. [Bhatt et al. 2017; Bhatt 2018] to create an implementation of HGABAC utilizing the NIST Policy Machine (PM) [Ferraiolo et al. 2011, 2015]. To accomplish this feat, a more restricted model of HGABAC, entitled Restricted Hierarchical Group and Attribute-Based Access Control (rHGABAC), was created and formalized as a single-value enumerated policy. Due to this single-value enumeration (vs. the enumerated policies used in HGABAC and other models), rHGABAC is unable to represent all policies possible in the full HGABAC model (i.e. rHGABAC is unable to represent conjunctive policies). While limiting, this restriction is advantageous for use with the PM as it is also a single-value enumerated authorization policy. Using rHGABAC, Bhatt et al. were able to configure the PM for HGABAC and compare policy evolution times with a Role-Centric ABAC configuration and show that HGABAC had comparable times.

7.1.2 Attribute Groups and Hierarchies

In addition to extensions to HGABAC, numerous works have adopted or explored using the novel concept of hierarchical user and object attribute groups HGABAC introduced into their own ABAC models and frameworks. A recent publication of note is Fernández et al.'s [Fernández et al. 2019; Fernández and Thuraisingham 2018] proposed *"pure"*[1] ABAC model, abbreviated as *C-ABAC*, based on the notion of category from Barker's 2009 unifying access control meta-model [Barker 2009]. In *C-ABAC*, entities (users, objects, etc.) are assigned to categories based on their attributes and categories of entities, as a whole, are assigned permissions. While similar to HGABAC's notation of groups, categories are distinct in that assignment to a category is based on the entities attributes as opposed to being manually assigned. Fernández et al. also define a version of C-ABAC that supports HGABAC style hierarchical groups based on a restricted version of HGABAC that lacks sessions [Fernández et al. 2019].

Several recent *"domain specific"*[2] ABAC models have also included the HGABAC concept of attribute groups and hierarchies. Gupta et al. [Gupta et al. 2019a,b] present an ABAC model targeted at next-generation smart cars referred to as CV-ABAC$_\mathrm{G}$ (*"CV"* presumably standing for *"connected vehicles"*). CV-ABAC$_\mathrm{G}$ is a hierarchical group based model that uses physical location groups that are dynamically assigned to vehicles based on their current location, attributes and/or personal preferences. Location groups are hierarchical, with child groups inheriting the attributes of parent groups (as is standard in HGABAC groups).

A second but distinct effort, also by Gupta et al. [Gupta et al. 2018], aims to create an ABAC model for big data processing in the Hadoop framework[3]. Gupta et al. put forth an ABAC model, entitled *HeABAC*, for securing a multi-tenant Hadoop ecosystem that may be implemented using Apache Ranger. *HeABAC* is an evolution of *OT-RBAC* [Gupta et al. 2017], an earlier RBAC based Hadoop focused access control model. Both *HeABAC* and *OT-RBAC* incorporate hierarchical user groups directly inspired by HGABAC that support inheritance from parent groups. In *OT-RBAC*, this takes to form of user groups that are assigned and inherit roles (which in turn grant permissions in the manner of RBAC). In *HeABAC*, user groups function more similarly to HGABAC groups, that is, they are assigned and inherit user and subject attributes (subject attributes being distinct from user attributes in the *HeABAC* model). Both models are proposed to be implementable as plug-ins for Apache Ranger[4] but actual implementation is left to future work.

Beyond the scope of ABAC models is Manar et al.'s [Alohaly et al. 2019] approach to mining Natural Language Access Control Policy (NLACP) into machine readable ABAC policies

[1]As defined in Chapter 2 Section 2.2.
[2]See Footnote 1.
[3]`https://hadoop.apache.org/`
[4]`https://ranger.apache.org/`

using machine learning and natural language processing techniques. HGABAC is chosen by Manar et al. as the reference ABAC model for this endeavour as its hierarchical structure better reflects the requirements of real-world organizations. HGABAC user and object groups are used as a hierarchical namespace for classifying attributes mined from policies.

7.1.3 Implementation of Delegation Strategies

The delegation strategies laid out in Chapter 4 (published as [Servos and Osborn 2016]) describe a number of approaches for incorporating delegation in modern ABAC models. While each strategy is categorized, detailed and the trade-off discussed, the actual formalization and implementation of each strategy into a working model is left to future work. To date, at least one work (excluding our own effort detailed in Chapter 6) by others has taken the blueprints developed in Chapter 4 and created a their own attribute-based delegation model. Sabahein et al. [Sabahein et al. 2018] extend the HGABAC model to support *User-to-Attribute Group Membership Delegation* as described in Chapter 4 Section 4.2.2.2 with an end goal of managing information sharing in a cloud based healthcare information system. A formal revocation system is developed to supplement the delegation model which supports both revocation initiated by a delegator and as a result of its lifetime expiring. Options are given to support local or global propagation (non-cascading or cascading revocations of subsequent delegations) as well as deal with conflicts arising from delegations from multiple delegators to the same delegatee (i.e. dominance). Finally, a XACML based architecture and new ABAC specification language are proposed to support their extended HGABAC model. Problems with conflicting policy evaluations and user collusion identified as key issues with *User-to-Attribute Group Membership Delegation* in Chapter 4 are left to future work by the authors.

The work in Chapter 4 has also aided in brining to light a number of possible issues with delegation in attribute-based and other next generation access control models that need to be addressed by researchers creating new and novel delegation models. The issue of user conclusion (in which two or more users could collude to create a delegation with more permission than either user could achieve individually) for instance is already being discussed and avoided in other works [Al-Wahah and Farkas 2018; Sabahein et al. 2018] to some extent based on its description in [Servos and Osborn 2016].

7.2 Concluding Remarks

In Chapter 1 Section 1.3.1, a number of goals for this research were outlined in addition to the more general goal of solving a number of the open problems identified in our survey of current ABAC literature (Chapter 2 Section 2.4). These goals included; 1. creating a hierarchical ABAC model, 2. representing the traditional models (MAC, DAC, RBAC) in ABAC, 3. ensuring support for distributed systems, 4. providing a supporting architecture to *"fill in the gaps"*, and 5. incorporating delegation into ABAC.

To satisfy these goals a number of contributions were developed and published to aid in the wider adoption of ABAC. These contributions are summarized in the following subsections (Sections 7.2.1 to 7.2.5) and details are given to show how they work towards fulfilling each

Table 7.1: Major Contributions vs. Research Goals. Check marks (✓) indicate that this contribution aided in satisfying the corresponding goal.

	HGABAC (Chapter 3)	**HGAA** (Chapter 5)	**Delegation Strategies** (Chapter 4)	**User-To-User Attribute Delegation** (Chapter 6)
Hierarchical ABAC Model	✓			
Representing Traditional Models	✓			
Support for Distributed Systems		✓		✓
Supporting Architecture		✓		✓
Delegation			✓	✓

stated goal. Table 7.1 summarizes the contribution of each chapter (excluding Chapter 2 from which the goals are in part derived) towards a corresponding research goal.

7.2.1 Survey and Taxonomy of ABAC Models

When conducting the literature review in Chapter 2, few if any, works had sought to provide a detailed survey of the current ABAC related research. The main contribution of Chapter 2 was providing such a survey as well as the identification of open problems (Chapter 2 Section 2.4) and gaps in the literature at the time. The problems identified aided in setting the direction and goals for the research contained in this thesis as well as outlining directions for future work in the field as a whole that needs to be addressed for ABAC to become a widely accepted model of access control.

The taxonomy provided as part of the survey (Chapter 2 2.2), categorizes the body of ABAC research into hierarchical subcategories to ease discussion and comparisons of closely related works. Of particular note is the categorization of ABAC models into *"pure"* models, that are not extensions to existing or traditional models, and *"hybrid"* models, which add attribute-based features to existing access control models. *"Pure"* are further subdivided into *"general"* and *"domain specific"* based on if the model is designed for a specific application or for general access control use.

A key finding of the review was that ABAC publications (in particular new models and formalizations of ABAC) had been steadily increasing since 2005 up to at least the time the review was completed in 2014 (Chapter 2 Section 2.1). Despite this clear interest, there was still little standardization and many critical access control aspects like delegation, administration, hierarchical structures, SoD, etc. had been overlooked or left to future work. These issues had hindered acceptance of ABAC outside of academia and lead in part to the creation of HGABAC in an effort to provide solutions to at least a few of these roadblocks.

7.2.2 Hierarchical Group and Attribute-Based Access Control (HGABAC)

The creation of the HGABAC model was motivated by the lack of formalized general purpose ABAC models available at the time and sought to bring hierarchical group based representations to ABAC, something that was novel at the time and could greatly aid in reducing the complexity of administering user and object attributes. Secondary goals were to allow for new ways to represent the traditional models of access control in ABAC and provide a simplified but still flexible policy representations to both aid in user comprehension and make reasoning about the security of such policies easier. At the time, Jin et al. had recently published their ABAC$_\alpha$ [Jin et al. 2012b] model which included representations for MAC, DAC and RBAC based policies without the use of groups (instead of relying on attributes that held partial ordered sets of values). HGABAC showed that these policies could also be represented with hierarchical groups (Chapter 3 Section 3.5) and that fewer attribute assignments were needed in the case of most policies (Chapter 3 Section 3.4.2).

The *"administrative attribute"*, a new attribute type that acts as a system wide semi-permanent constant was also introduced by the HGABAC model. This attribute type allows for system administrators to set global variables that may affect all policies that include them regardless of the user or object involved. For example, a threat level administrative attribute may be created that changes the strictness of policies based on its value.

The policy language introduced by HGABAC (Chapter 3 3.3.2), now referred to as Hierarchical Group Policy Language (HGPL), provides a simplified language for specifying ternary policy statements based on Kleene K3 logic [Kleene 1938]. HGPL closely resembles C-style boolean statements, but with an additional possible result of *"UNDEF"* added to the traditional *"TRUE"* and *"FALSE"*. This resemblance aids both policy authors and users by putting access control policies in a more straightforward familiar form. Hierarchical Group Policy Language (HGPL) was further updated in Chapter 5 5.4.1, as HGPLv2, to include support for attribute namespaces and improvements to aid parsing by the HGPLv2 interpreter.

Since the publication of [Servos and Osborn 2014], the concept of hierarchical attribute groups introduced by HGABAC has seen some acceptance and impact in the ABAC literature (as described in Section 7.1) primarily for its simplification of attribute administration. While this core contribution provided a formal model of ABAC, it was not enough on its own to base a full access control system on. There was a clear need for a supporting architecture to fill in the gaps between a model and implementable system. This need lead to the development of the supporting HGAA architecture.

7.2.3 Hierarchical Group Attribute Architecture (HGAA)

The HGAA architecture presented in Chapter 5 provides solutions to a number of implementation questions left unanswered by the HGABAC model on its own. Questions like *"what service assigns the attributes?"*, *"how are attributes shared with each party?"*, *"how does the* *user provide proof of attribute ownership?"*, and *"where and how are policies evaluated?"* resolved through four key contributions; the HGAA architecture and protocol, a new Attribute Certificate (AC) format, a URI based attribute namespace and a HGPLv2 policy language interpreter.

The HGAA architecture details the services required for the sharing of attributes and users between isolated security domains and explains how access control decisions can be made independently of the users identity and without directly connecting to their home domain. A high level protocol is documented that explains what information is exchanged between each service. Particular effort was taken to ensure the functionality of the architecture in distributed environments (an ongoing goal of this research) and that users could be authenticated in an *"off-line"* manner (i.e. without having to directly request information from their home domain or third party).

The *"off-line"* authentication and interoperability between independent security domains is made possible in HGAA in part by the concept of the Attribute Certificate (AC) introduced in Chapter 5 Section 5.4.3. The AC format provides a cryptography secure document for sharing a user's attributes (including connection attributes) with third party services. Revocation rules are included in the AC to invalidate the certificate after set rules are triggered (limited to an expiry date and revocation list in the first version of the AC format). A number of points for extensions were intentionally included in the AC to allow for the future work in Chapter 6 to support delegation as well as future use in other research projects.

Updates to the HGPL policy language including a namespace for attributes and other HGABAC elements were introduced (Chapter 5 Sections 5.4.1 and 5.4.5) to allow them to be uniquely identifiable across independent organizations and further increase the flexibility of the policy language. A prototype interpreter for the HGPLv2 language was implemented and a number of optimization steps were suggested to improve policy evaluation time. Preliminary results from evaluating both the prototype interpreter and HGAA services were promising (Chapter 5 5.5), showing that AC size and time to generate grew linearly with the number of user attributes it contained. Time for services to process and execute requests also grew linearly with the number of user attributes and a similar linear growth pattern was found for the policy language interpreter based on the number of AST nodes a policy contained. These findings suggest that a full real-world implementation of HGAA would be reasonably scalable in most settings, including distributed environments.

7.2.4 Delegation Strategies

The delegation strategies put forth in Chapter 4 speculate about the feasible ways in which delegation can be incorporated into ABAC models with attribute group support such as HGABAC. These strategies, developed by appraising each combination of three access control components; a delegator, a delegatee and a delegatable access control element, are categorized into three families (Attribute Delegation, Group Membership Delegation and Permission Delegation) based on the access control element being delegated. Each family of strategies is evaluated for advantages, disadvantage and potential security issues (e.g. possibility for user collusion). The result of this evaluation (summarized in Chapter 4 4.4.1) provides a useful contribution as a guideline for the selection of delegation strategy to be utilized when creating a new ABAC delegation model. As no one strategy is ideal in all cases, such a guideline is important for understanding the trade-offs associated with each method of implementing delegation and the potential security issues associated with it.

Beyond leading to the creation of the User-to-User Attribute Delegation model formalized in Chapter 6, the strategies developed in Chapter 4 have already had some impact on the development of ABAC (and other) delegation models by other researchers as discussed in Section 7.1.3. It is anticipated these strategies will continue to play a role in guiding the development of future ABAC delegation models and have at very least identified possible attribute-based delegation models that have not yet been formalized or explored in the current literature.

7.2.5 User-to-User Attribute Delegation

The last major contribution is the User-to-User Attribute Delegation model for HGABAC introduced in Chapter 6 which consists of an extension to HGABAC as well as updates to HGAA (including the AC format). These extensions implement and formalize the User-to-User Attribute Delegation strategy from Chapter 4 into a new and novel attribute-based delegation model, satisfying the goal of bringing delegation to ABAC.

Changes to the AC format take advantage of places left for future expansion and delegation extensions as outlined in Chapter 5 5.4.3. This update to the AC allows for User-to-User delegation to be performed in an *"off-line"* manner, that is, without connecting or communicating with a third party. After users are issued an AC from an attribute authority, they may delegate attributes marked as delegatable to another user without connecting to any service or communicating with any actor other than the delegatee by issuing an Discretionary Access Control (DAC). This is a critical feature as it further aids in supporting distributed systems (in which direct communication with the issuing attribute authority may not be possible or prohibitively costly) and distinguishes this work from other attempts at using the delegation strategies from Chapter 4 to create a delegation model for HGABAC such as that by Sabahein et al. [Sabahein et al. 2018] (previously discussed in Section 7.1.3).

7.3 Directions for Future Work

A number of directions for future work have been identified during the course of this research, both for the area of ABAC in general and specific to the proposed models, frameworks and architectures presented in this thesis. The following subsection summarize the possible directions identified in each chapter (Chapters 2 to 6) and briefly discuss recommendations for research towards potential solutions.

7.3.1 Surveying ABAC

The literature survey in Chapter 2 offered one of the first large scale reviews of ABAC related research, however, a number of areas were left unexplored or out of the scope of the work. The survey focusses on covering a wide range of models in breadth, leaving room for further survey works to explore specific categories or aspects of ABAC models in-depth. In particular, an analysis of how current models represent attribute-based policies or an in-depth review of specific subcategories of models (e.g. a closer look at Pure General ABAC models) would be beneficial to the community.

As the focus of the survey was models and access control frameworks, reviews of non-model related attribute topics such as attribute & policy mining, attribute storage & sharing, attribute confidentiality & privacy, attribute-based policy languages, and supporting model independent architectures could also be of interest and open to further surveying.

Since the publication of the survey as [Servos and Osborn 2017], a number of new ABAC efforts have been released. An update to the survey to include more recent research would help to keep up with the rapid developments in the field.

Chapter 2 also identified a number of open problems that were out of the scope of the research contained in this thesis. These are outlined in depth in Chapter 2 Section 2.4 but include issues such as; Auditability (Chapter 2 Section 2.4.4), Separation of Duties (Chapter 2 Section 2.4.5), Administration (Chapter 2 Section 2.4.9), and Formal Security Analysis (Chapter 2 Section 2.4.10). While not addressed directly by this research, solving these issues are important for making ABAC more usable, secure and aiding in more wide scale acceptance outside of academia.

7.3.2 Hierarchical Group and Attribute-Based Access Control (HGABAC)

HGABAC successfully formalizes a model of ABAC with hierarchical group support and later extensions add User-to-User delegation. However, there are still a number of areas for potential improvement or exploration. Perhaps the most important direction for future extension to model is adding support for Separation of Duties (SoD). SoD is the concept of segregating users effective permissions such that more than one person is required to complete a task to help prevent fraud, error or conflicts of interest. Currently, HGABAC has no explicit support for SoD, although some limited form may be possible through complex HGPL policies. Work is needed to show that HGPL is flexible enough to enforce SoD constraints and/or to update HGPL and HGABAC to directly support SoD constraints. A number of recent works have sought to introduce SoD to ABAC, including the constraint specification language by Bijon et al. [Bijon et al. 2013], the fromalization of SoD constraints for ABAC components by Jha et al. [Jha et al. 2017], and the AHCSABAC model by Singh [Singh 2016] which includes SoD constraints. More research is needed to see if these could be incorporated with the HGABAC model or HGPL.

A second issue not address by HGABAC, is the administration of users, attributes, permissions and policies. HGAA lays out where administrative services would fit in the system's architecture and how they would exchange information, but an actual administration model and implementation of administrative services was left for future work. Some recent efforts have put forth a start at extending HGABAC with an administration model, such as GURA_G [Gupta and Sandhu 2016], however much more needs to be done before a full implementation is possible (such as supporting object attribute groups in GURA_G).

The HGPL policy language introduced with the HGABAC model and updated in Chapter 5 (to support attribute namespaces and HGAA) provides a good first effort towards a simplified but flexible general use attribute-based policy language for both academic and real world use, however, there are still a number of areas that could be enhanced. As previously mentioned, SoD constraints are needed to enforce the concept of separation of duties and aid in better representing RBAC policies (that may contain SoD constraints). Exploring using XACML and other existing generic policy languages with HGABAC, in place of HGPL, could lead to a better understanding of both the performance trade-offs, flexibility and administrator/user comprehension benefits of HGPL based policies.

While there has been a significant push for the adoption and development of ABAC[Council and Architecture 2011; of the Press Secretary 2012; Hu et al. 2013], many systems and organizations are still using access control systems and policies based on the traditional models (namely RBAC). Work will be needed to study how ABAC models such as HGABAC can be used in conjunction with existing legacy RBAC systems or how the migration to next generation access control systems can be automated and simplified. A possible step in this direction, are the hybrid models discussed in Chapter 2 Section 2.3.2 which add attributes and rule-based policy concepts as extensions to the traditional models. Many of these hybrid models follow one or more of the possible strategies for adding attributes to RBAC outlined and discussed by Kuhn et al. [Kuhn et al. 2010]. Another direction would be providing tools to map RBAC policies to HGABAC policies and group hierarchies which are capable of encapsulating and enforcing RBAC based policies (as described in Chapter 3 Section 3.5.3).

The addition of conditional user and object groups, in which users (or objects) are automatically added to groups based on set policies is another interesting possible future direction for extending HGABAC. Such automation could further simplify administration and have interesting implications that are worthy of future research. A number of works, such as the *C-ABAC* model [Fernández et al. 2019; Fernández and Thuraisingham 2018], have already made some progress in this direction (as discussed in Section 7.1.2) but it remains to be seen if the same technique would be appropriate and useful for HGABAC.

Lastly, a full reference implementation of HGABAC and HGAA is needed to aid in *"real world"* acceptance and use outside of academia. Current efforts to date have centered around *"research grade"* implementation of specific components for the purposes of evaluation and comparison with other models/architectures and are not yet suitable for secure *"real world"* use.

7.3.3 Hierarchical Group Attribute Architecture (HGAA)

The HGAA architecture introduced in Chapter 5 intentionally leaves room for future extensions and efforts to improve the architecture. One such extension is the work in Chapter 6 that adds User-to-User delegation to HGABAC and HGAA. However, there is still room for further extensions. The Attribute Certificate (AC) format described in Chapter 5 Section 5.4.3 allows for certificate revocation via an expiry date or revocation list, but room is left for future extensions to add more complex revocation rules. Work is needed to explore if using HGPL based revocation rules based on user, connection, environment, or other attribute values could provide a more flexible, scalable and robust AC revocation system.

Creating a version of HGAA using existing generic access control standards such as XACML, SAML, Kerberos[Neuman and Ts'o 1994], and/or pure X.509 attribute certificates would allow for a more in-depth comparison of the performance and flexibility of HGAA vs. what is currently available. Utilizing XACML and SAML could also lead to greater compatibility with existing system and aid acceptance in the *"real world"*.

Further work is also needed towards testing the utility and effectiveness of HGAA in nontraditional computing paradigms including Internet of Things (IoT), embedded systems, use with physical smart identity cards, smart connected vehicles and cloud computing. In some cases, alternative architectures for HGABAC may be more appropriate or have distinct advantages for particular use cases. It is likely that a number of *"domain specific"* architectures or extensions to HGAA will be needed to handle a number of these nontraditional paradigms.

Finally, HGAA has a number of future directions in common with HGABAC, including the need for administration services (currently left unimplemented), the need for a reference implantation for secure *"real world"* use, and a more formal security analysis of the framework.

7.3.4 Delegation

The main long term direction for delegation in HGABAC and HGAA, is the formalization and implementation of each delegation strategy in Chapter 4. To date, User-to-User Attribute Delegation (Chapter 6) and User-to-Attribute Group Membership Delegation [Sabahein et al. 2018] have been formalized but the remainder are only described informally. Working implementations of each strategy would allow for more comprehensive and quantitative evaluations. Research exploring combining or utilizing multiple delegation strategies simultaneously could also provide new possibilities for delegation that might aid in overcoming limitations of individual strategies but risk introducing new conflicts and security issues.

In terms of the User-to-User Attribute Delegation model described in Chapter 6, replacing the *"Can Delegate"* relation for designating what attributes users can delegate with a *"Can Receive"* style relationship may lead to a more flexible delegation model. Such a relation has been used successfully for RBAC [Crampton and Khambhammettu 2008] and is used in Sabahein et al. [Sabahein et al. 2018] User-to-Attribute Group Membership Delegation model but work is needed to explore its potential for User-to-User Attribute Delegation.

Bibliography

Mouiad Al-Wahah and Csilla Farkas. Context delegation for context-based access control. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 201–210. Springer, 2018.

Manar Alohaly, Hassan Takabi, and Eduardo Blanco. Automated extraction of attributes from natural language attribute-based access control (ABAC) policies. *Cybersecurity*, 2(1):2, 2019.

Steve Barker. The next 700 access control models or a unifying meta-model? In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 187–196. ACM, 2009.

Smriti Bhatt. *Attribute-Based Access and Communication Control Models for Cloud and Cloud-Enabled Internet of Things*. PhD thesis, UNIVERSITY OF TEXAS AT SAN ANTONIO, 2018.

Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. ABAC with group attributes and attribute hierarchies utilizing the policy machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control*, pages 17–28. ACM, 2017.

Khalid Zaman Bijon, Ram Krishman, and Ravi Sandhu. Constraints specification in attribute based access control. *Science*, 2(3):131, 2013.

Federal Chief Information Officers Council and Federal Enterprise Architecture. Federal identity, credential, and access management (FICAM) roadmap and implementation guidance version 2.0. Technical report, CIO Council, December 2011. URL https://www.idmanagement.gov/wp-content/uploads/sites/1171/uploads/FICAM_Roadmap_and_Implem_Guid.pdf.

Jason Crampton and Hemanth Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2):123–136, 2008.

Maribel Fernández and Bhavani Thuraisingham. A category-based model for ABAC. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 32–34. ACM, 2018.

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. ACM, 2019.

David Ferraiolo, Vijayalakshmi Atluri, and Serban Gavrila. The policy machine: A novel architecture and framework for access control policy specification and enforcement. *Journal of Systems Architecture*, 57(4):412–424, 2011.

David F Ferraiolo, Serban I Gavrila, and Wayne Jansen. Policy machine: features, architecture, and specification. Technical report, 2015.

Maanak Gupta and Ravi Sandhu. The GURA$_G$ administrative model for user and group attribute assignment. In *International Conference on Network and System Security*, pages 318–332. Springer, 2016.

Maanak Gupta, Farhan Patwa, and Ravi Sandhu. Object-tagged RBAC model for the hadoop ecosystem. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 63–81. Springer, 2017.

Maanak Gupta, Farhan Patwa, and Ravi Sandhu. An attribute-based access control model for secure big data processing in Hadoop ecosystem. In *Proceedings of the Third ACM Workshop on Attribute-Based Access Control*, pages 13–24. ACM, 2018.

Maanak Gupta, James Benson, Farhan Patwa, and Ravi Sandhu. Dynamic groups and attribute-based access control for next-generation smart cars. In *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, pages 61–72. ACM, 2019a.

Maanak Gupta, James Benson, Farhan Patwa, and Ravi Sandhu. Secure cloud assisted smart cars using dynamic groups and attribute based access control. *arXiv preprint arXiv:1908.08112*, 2019b.

Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST Special Publication*, 800:162, 2013.

Sadhana Jha, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya. Specification and verification of separation of duty constraints in attribute-based access control. *IEEE Transactions on Information Forensics and Security*, 13(4):897–911, 2017.

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

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, 2012b.

Stephen Cole Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4): 150–155, 1938.

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

B Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications magazine*, 32(9):33–38, 1994.

Office of the Press Secretary. National strategy for information sharing and safeguarding. http://www.whitehouse.gov/the-press-office/2012/12/19/national-strategy-information-sharing-and-safeguarding, December 2012. Accessed: 2014-12-07.

Khaled Sabahein, Brian Reithel, and Feng Wang. Incorporating delegation into ABAC: Healthcare information system use case. In *Proceedings of the International Conference on Security and Management (SAM)*, pages 291–297. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2018.

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.

Daniel Servos and Sylvia L Osborn. HGABAC: Towards a formal model of hierarchical attribute-based access control. In *The 7th International Symposium on Foundations and Practice of Security (FPS'2014)*, pages 187–204, November 2014.

Daniel Servos and Sylvia L Osborn. Strategies for incorporating delegation into attribute-based access control (ABAC). In *The 9th International Symposium on Foundations and Practice of Security (FPS'2016)*, pages 320–328, October 2016.

Daniel Servos and Sylvia L Osborn. Current research and open problems in attribute-based access control. *ACM Computing Surveys (CSUR)*, 49(4):65, 2017.

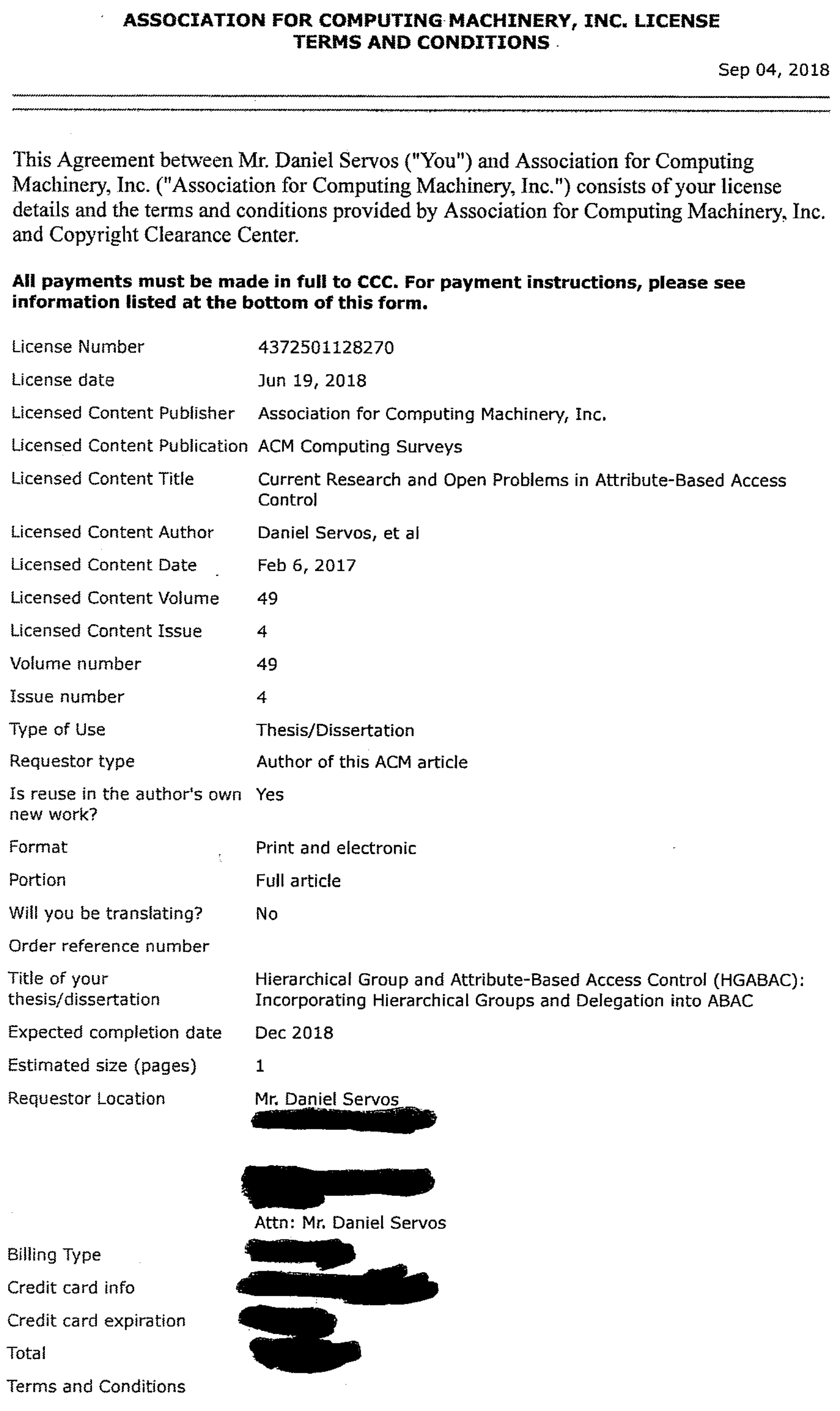
Mahendra Pratap Singh. AHCSABAC: Attribute value hierarchies and constraints specification in attribute-based access control. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 35–41. IEEE, 2016.

```
  _
>(.)__
 (___/
```

Appendix A

Copyright Permissions

A.1 Chapter 2: Current Research and Open Problems in Attribute-Based Access Control



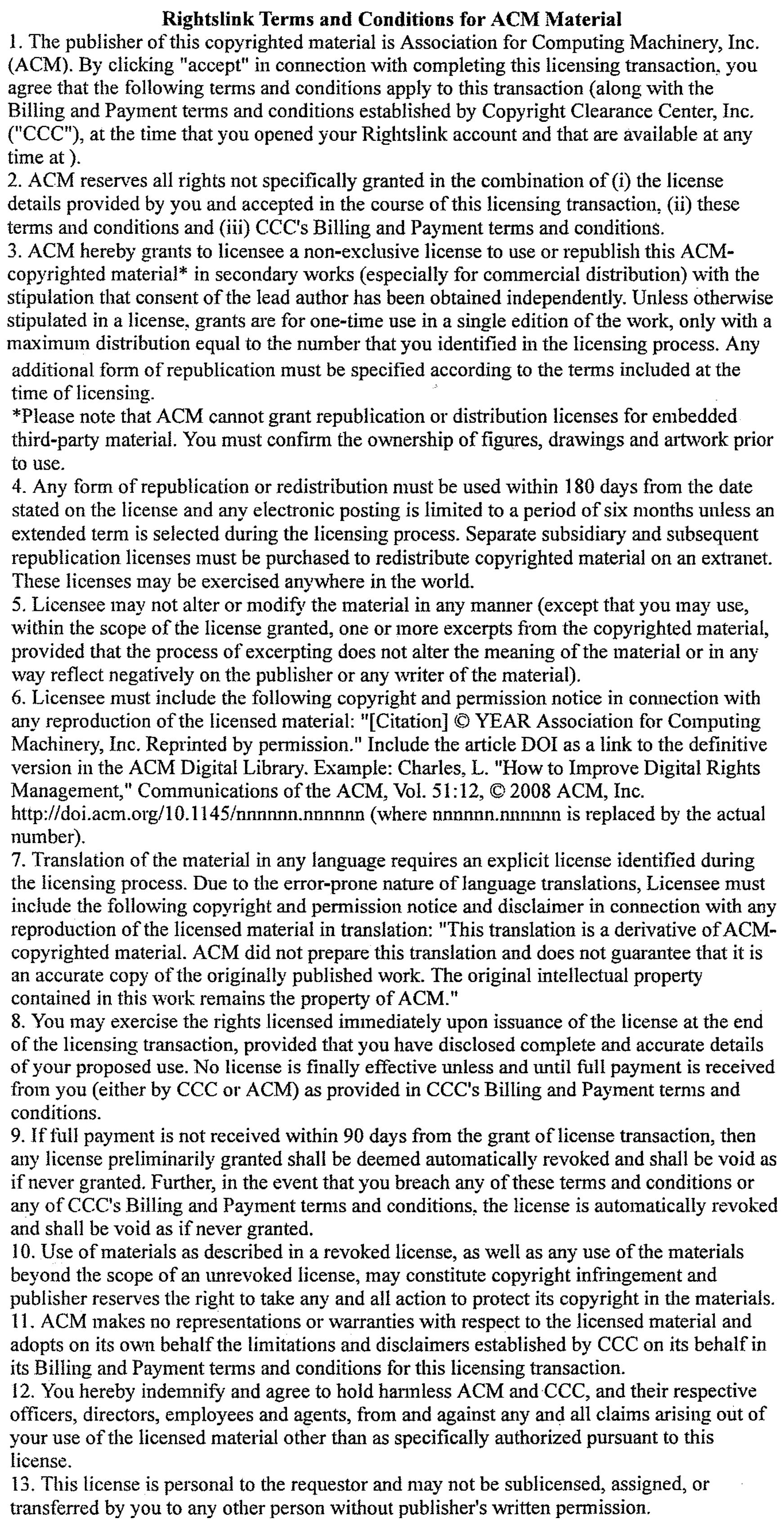
ASSOCIATION FOR COMPUTING MACHINERY, INC. LICENSE TERMS AND CONDITIONS

Sep 04, 2018

This Agreement between Mr. Daniel Servos ("You") and Association for Computing Machinery, Inc. ("Association for Computing Machinery, Inc.") consists of your license details and the terms and conditions provided by Association for Computing Machinery, Inc. and Copyright Clearance Center.

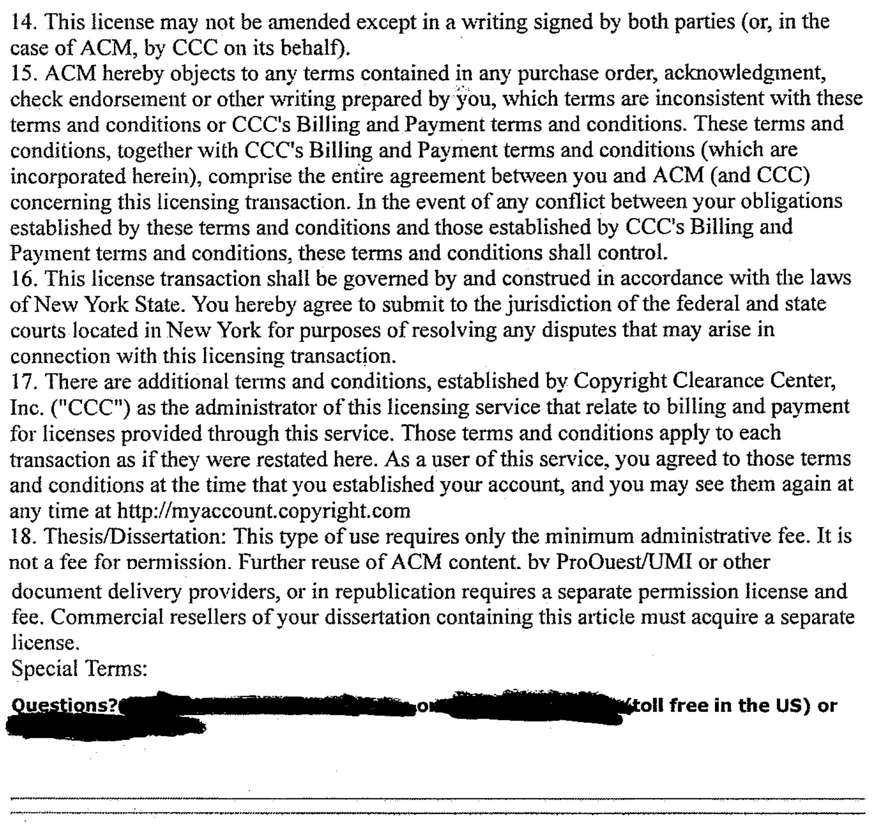
All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	4372501128270
License date	Jun 19, 2018
Licensed Content Publisher	Association for Computing Machinery, Inc.
Licensed Content Publication	ACM Computing Surveys
Licensed Content Title	Current Research and Open Problems in Attribute-Based Access Control
Licensed Content Author	Daniel Servos, et al
Licensed Content Date	Feb 6, 2017
Licensed Content Volume	49
Licensed Content Issue	4
Volume number	49
Issue number	4
Type of Use	Thesis/Dissertation
Requestor type	Author of this ACM article
Is reuse in the author's own new work?	Yes
Format	Print and electronic
Portion	Full article
Will you be translating?	No
Order reference number	
Title of your thesis/dissertation	Hierarchical Group and Attribute-Based Access Control (HGABAC): Incorporating Hierarchical Groups and Delegation into ABAC
Expected completion date	Dec 2018
Estimated size (pages)	1
Requestor Location	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Billing Type	[redacted]
Credit card info	[redacted]
Credit card expiration	[redacted]
Total	[redacted]
Terms and Conditions	



Rightslink Terms and Conditions for ACM Material

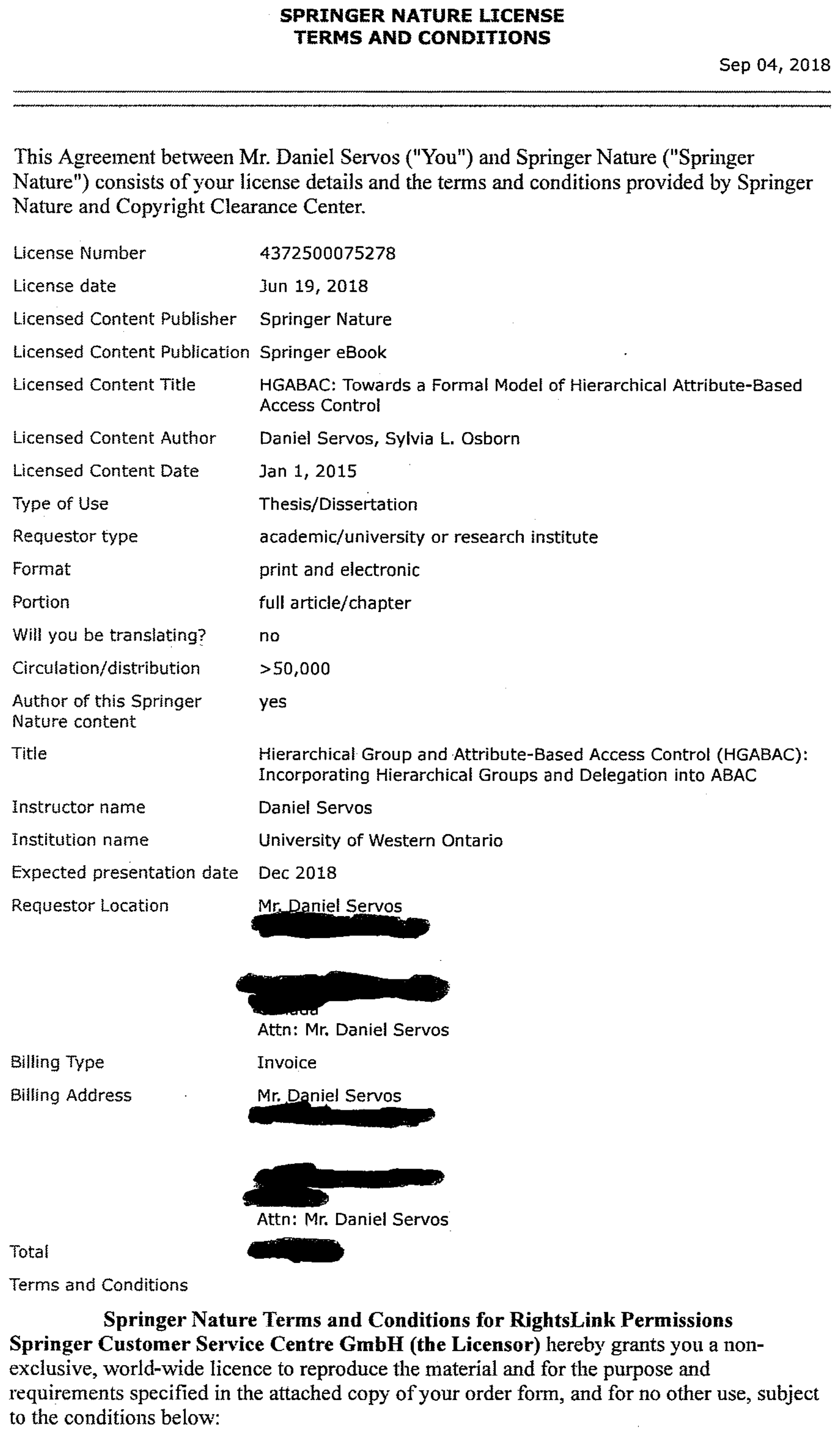
1. The publisher of this copyrighted material is Association for Computing Machinery, Inc. (ACM). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at).
2. ACM reserves all rights not specifically granted in the combination of (i) the license details provided by you and accepted in the course of this licensing transaction, (ii) these terms and conditions and (iii) CCC's Billing and Payment terms and conditions.
3. ACM hereby grants to licensee a non-exclusive license to use or republish this ACM-copyrighted material* in secondary works (especially for commercial distribution) with the stipulation that consent of the lead author has been obtained independently. Unless otherwise stipulated in a license, grants are for one-time use in a single edition of the work, only with a maximum distribution equal to the number that you identified in the licensing process. Any additional form of republication must be specified according to the terms included at the time of licensing.
*Please note that ACM cannot grant republication or distribution licenses for embedded third-party material. You must confirm the ownership of figures, drawings and artwork prior to use.
4. Any form of republication or redistribution must be used within 180 days from the date stated on the license and any electronic posting is limited to a period of six months unless an extended term is selected during the licensing process. Separate subsidiary and subsequent republication licenses must be purchased to redistribute copyrighted material on an extranet. These licenses may be exercised anywhere in the world.
5. Licensee may not alter or modify the material in any manner (except that you may use, within the scope of the license granted, one or more excerpts from the copyrighted material, provided that the process of excerpting does not alter the meaning of the material or in any way reflect negatively on the publisher or any writer of the material).
6. Licensee must include the following copyright and permission notice in connection with any reproduction of the licensed material: "[Citation] © YEAR Association for Computing Machinery, Inc. Reprinted by permission." Include the article DOI as a link to the definitive version in the ACM Digital Library. Example: Charles, L. "How to Improve Digital Rights Management," Communications of the ACM, Vol. 51:12, © 2008 ACM, Inc. http://doi.acm.org/10.1145/nnnnnn.nnnnnn (where nnnnnn.nnnnnn is replaced by the actual number).
7. Translation of the material in any language requires an explicit license identified during the licensing process. Due to the error-prone nature of language translations, Licensee must include the following copyright and permission notice and disclaimer in connection with any reproduction of the licensed material in translation: "This translation is a derivative of ACM-copyrighted material. ACM did not prepare this translation and does not guarantee that it is an accurate copy of the originally published work. The original intellectual property contained in this work remains the property of ACM."
8. You may exercise the rights licensed immediately upon issuance of the license at the end of the licensing transaction, provided that you have disclosed complete and accurate details of your proposed use. No license is finally effective unless and until full payment is received from you (either by CCC or ACM) as provided in CCC's Billing and Payment terms and conditions.
9. If full payment is not received within 90 days from the grant of license transaction, then any license preliminarily granted shall be deemed automatically revoked and shall be void as if never granted. Further, in the event that you breach any of these terms and conditions or any of CCC's Billing and Payment terms and conditions, the license is automatically revoked and shall be void as if never granted.
10. Use of materials as described in a revoked license, as well as any use of the materials beyond the scope of an unrevoked license, may constitute copyright infringement and publisher reserves the right to take any and all action to protect its copyright in the materials.
11. ACM makes no representations or warranties with respect to the licensed material and adopts on its own behalf the limitations and disclaimers established by CCC on its behalf in its Billing and Payment terms and conditions for this licensing transaction.
12. You hereby indemnify and agree to hold harmless ACM and CCC, and their respective officers, directors, employees and agents, from and against any and all claims arising out of your use of the licensed material other than as specifically authorized pursuant to this license.
13. This license is personal to the requestor and may not be sublicensed, assigned, or transferred by you to any other person without publisher's written permission.



14. This license may not be amended except in a writing signed by both parties (or, in the case of ACM, by CCC on its behalf).
15. ACM hereby objects to any terms contained in any purchase order, acknowledgment, check endorsement or other writing prepared by you, which terms are inconsistent with these terms and conditions or CCC's Billing and Payment terms and conditions. These terms and conditions, together with CCC's Billing and Payment terms and conditions (which are incorporated herein), comprise the entire agreement between you and ACM (and CCC) concerning this licensing transaction. In the event of any conflict between your obligations established by these terms and conditions and those established by CCC's Billing and Payment terms and conditions, these terms and conditions shall control.
16. This license transaction shall be governed by and construed in accordance with the laws of New York State. You hereby agree to submit to the jurisdiction of the federal and state courts located in New York for purposes of resolving any disputes that may arise in connection with this licensing transaction.
17. There are additional terms and conditions, established by Copyright Clearance Center, Inc. ("CCC") as the administrator of this licensing service that relate to billing and payment for licenses provided through this service. Those terms and conditions apply to each transaction as if they were restated here. As a user of this service, you agreed to those terms and conditions at the time that you established your account, and you may see them again at any time at http://myaccount.copyright.com
18. Thesis/Dissertation: This type of use requires only the minimum administrative fee. It is not a fee for permission. Further reuse of ACM content, by ProQuest/UMI or other document delivery providers, or in republication requires a separate permission license and fee. Commercial resellers of your dissertation containing this article must acquire a separate license.
Special Terms:

Questions?[redacted] o[redacted]toll free in the US) or [redacted]

A.2 Chapter 3: HGABAC: Towards a Formal Model of Hierarchical ABAC



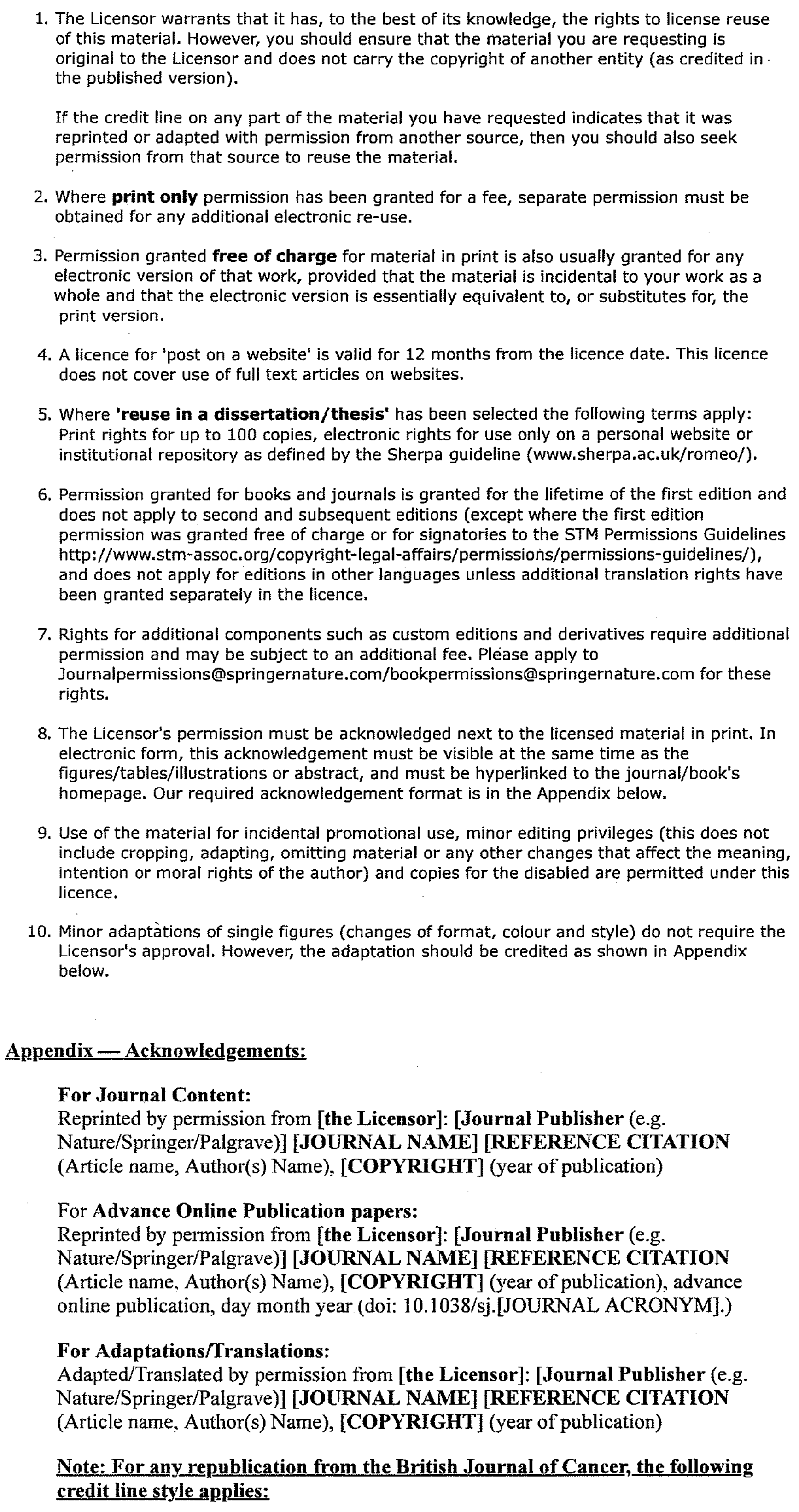
SPRINGER NATURE LICENSE
TERMS AND CONDITIONS

Sep 04, 2018

This Agreement between Mr. Daniel Servos ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	4372500075278
License date	Jun 19, 2018
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control
Licensed Content Author	Daniel Servos, Sylvia L. Osborn
Licensed Content Date	Jan 1, 2015
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	>50,000
Author of this Springer Nature content	yes
Title	Hierarchical Group and Attribute-Based Access Control (HGABAC): Incorporating Hierarchical Groups and Delegation into ABAC
Instructor name	Daniel Servos
Institution name	University of Western Ontario
Expected presentation date	Dec 2018
Requestor Location	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Billing Type	Invoice
Billing Address	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Total	[redacted]
Terms and Conditions	

Springer Nature Terms and Conditions for RightsLink Permissions
Springer Customer Service Centre GmbH (the Licensor) hereby grants you a non-exclusive, world-wide licence to reproduce the material and for the purpose and requirements specified in the attached copy of your order form, and for no other use, subject to the conditions below:



1. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of this material. However, you should ensure that the material you are requesting is original to the Licensor and does not carry the copyright of another entity (as credited in the published version).

   If the credit line on any part of the material you have requested indicates that it was reprinted or adapted with permission from another source, then you should also seek permission from that source to reuse the material.

2. Where **print only** permission has been granted for a fee, separate permission must be obtained for any additional electronic re-use.

3. Permission granted **free of charge** for material in print is also usually granted for any electronic version of that work, provided that the material is incidental to your work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.

4. A licence for 'post on a website' is valid for 12 months from the licence date. This licence does not cover use of full text articles on websites.

5. Where **'reuse in a dissertation/thesis'** has been selected the following terms apply: Print rights for up to 100 copies, electronic rights for use only on a personal website or institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/).

6. Permission granted for books and journals is granted for the lifetime of the first edition and does not apply to second and subsequent editions (except where the first edition permission was granted free of charge or for signatories to the STM Permissions Guidelines http://www.stm-assoc.org/copyright-legal-affairs/permissions/permissions-guidelines/), and does not apply for editions in other languages unless additional translation rights have been granted separately in the licence.

7. Rights for additional components such as custom editions and derivatives require additional permission and may be subject to an additional fee. Please apply to Journalpermissions@springernature.com/bookpermissions@springernature.com for these rights.

8. The Licensor's permission must be acknowledged next to the licensed material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract, and must be hyperlinked to the journal/book's homepage. Our required acknowledgement format is in the Appendix below.

9. Use of the material for incidental promotional use, minor editing privileges (this does not include cropping, adapting, omitting material or any other changes that affect the meaning, intention or moral rights of the author) and copies for the disabled are permitted under this licence.

10. Minor adaptations of single figures (changes of format, colour and style) do not require the Licensor's approval. However, the adaptation should be credited as shown in Appendix below.

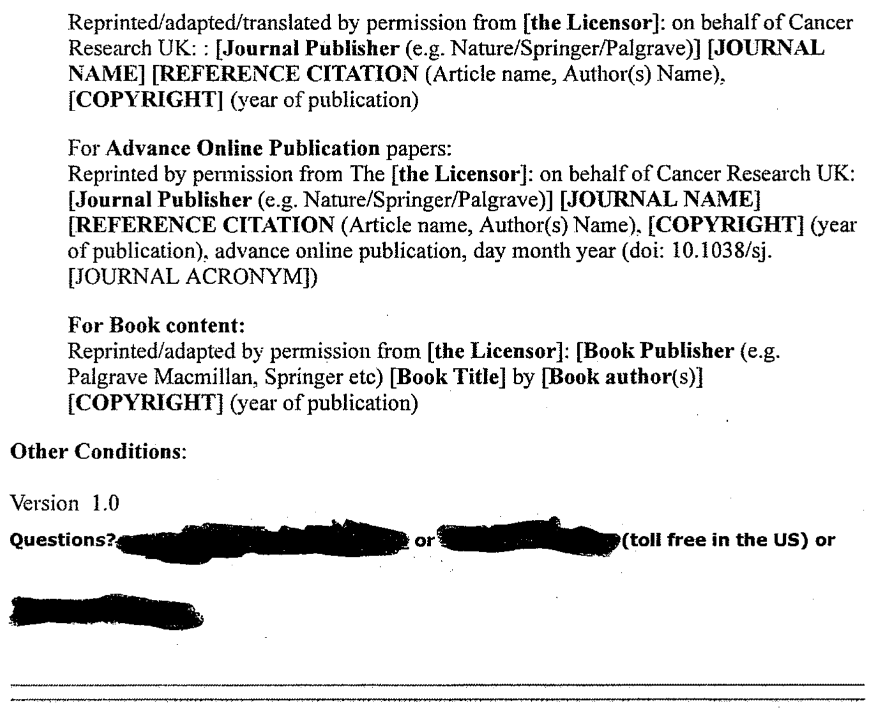
Appendix — Acknowledgements:

For Journal Content:
Reprinted by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication)

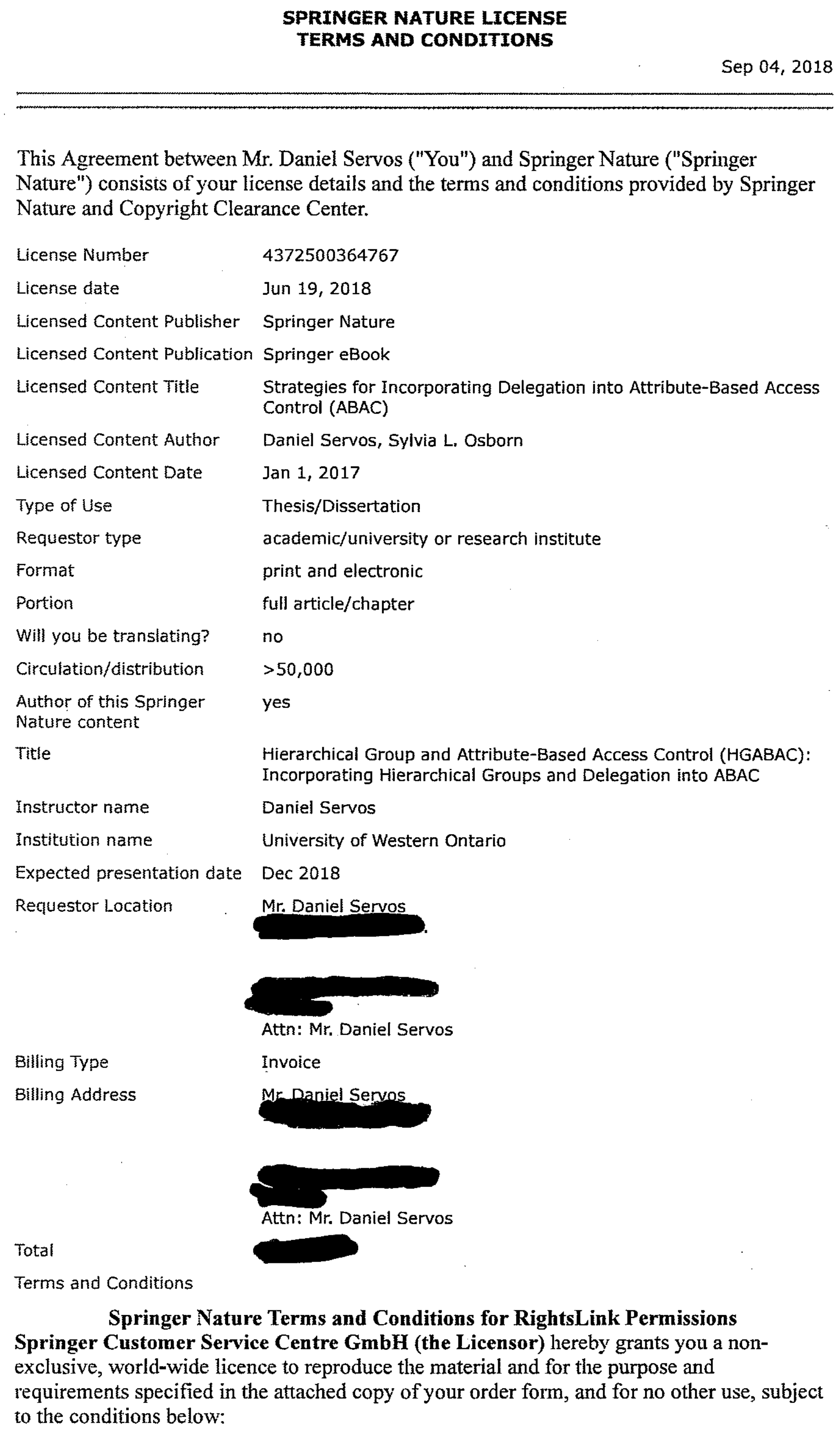
For **Advance Online Publication papers:**
Reprinted by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].)

For Adaptations/Translations:
Adapted/Translated by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication)

Note: For any republication from the British Journal of Cancer, the following credit line style applies:



A.3 Chapter 4: Strategies for Incorporating Delegation into ABAC



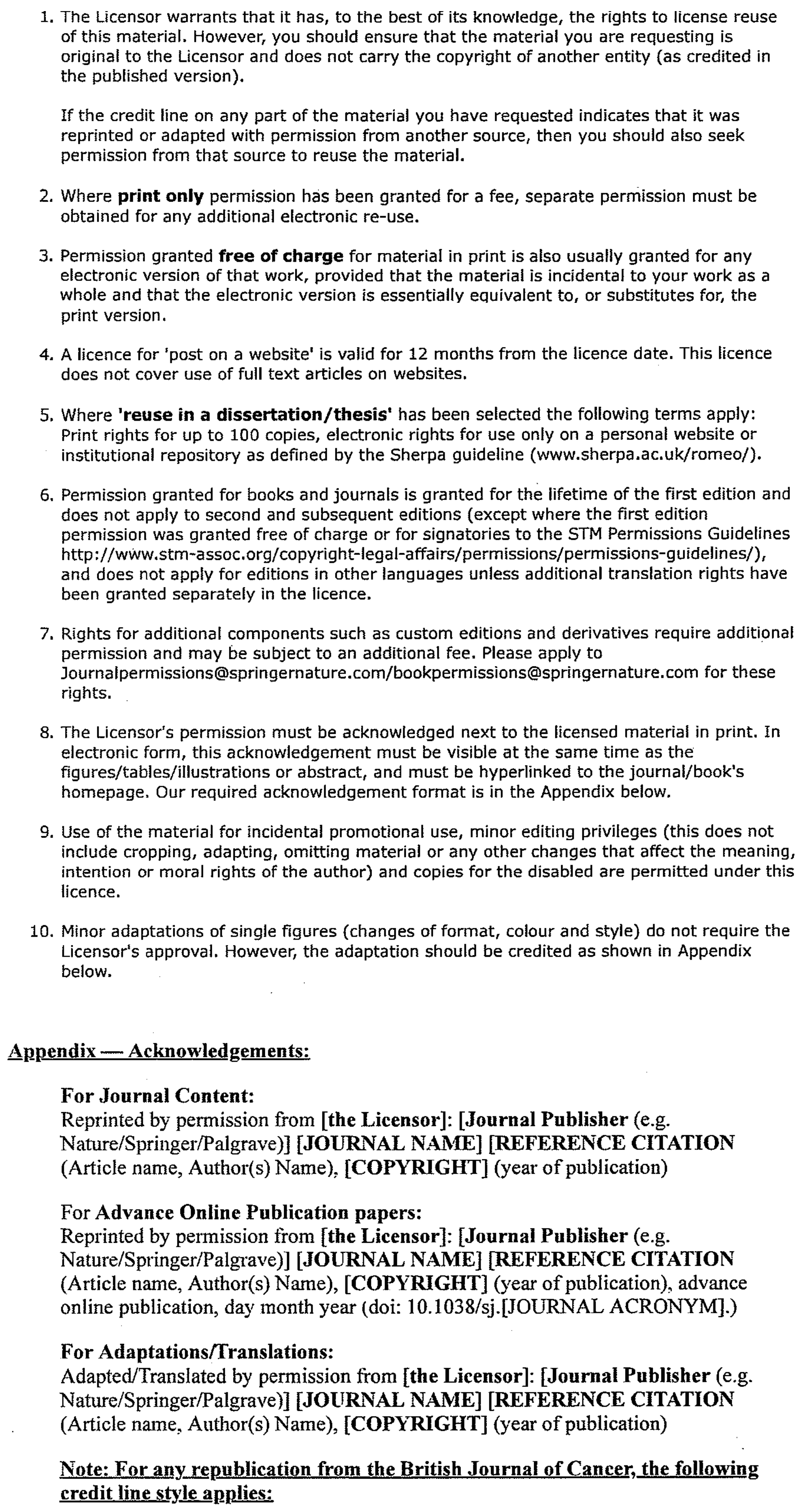
**SPRINGER NATURE LICENSE
TERMS AND CONDITIONS**

Sep 04, 2018

This Agreement between Mr. Daniel Servos ("You") and Springer Nature ("Springer Nature") consists of your license details and the terms and conditions provided by Springer Nature and Copyright Clearance Center.

License Number	4372500364767
License date	Jun 19, 2018
Licensed Content Publisher	Springer Nature
Licensed Content Publication	Springer eBook
Licensed Content Title	Strategies for Incorporating Delegation into Attribute-Based Access Control (ABAC)
Licensed Content Author	Daniel Servos, Sylvia L. Osborn
Licensed Content Date	Jan 1, 2017
Type of Use	Thesis/Dissertation
Requestor type	academic/university or research institute
Format	print and electronic
Portion	full article/chapter
Will you be translating?	no
Circulation/distribution	>50,000
Author of this Springer Nature content	yes
Title	Hierarchical Group and Attribute-Based Access Control (HGABAC): Incorporating Hierarchical Groups and Delegation into ABAC
Instructor name	Daniel Servos
Institution name	University of Western Ontario
Expected presentation date	Dec 2018
Requestor Location	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Billing Type	Invoice
Billing Address	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Total	[redacted]
Terms and Conditions	

Springer Nature Terms and Conditions for RightsLink Permissions
Springer Customer Service Centre GmbH (the Licensor) hereby grants you a non-exclusive, world-wide licence to reproduce the material and for the purpose and requirements specified in the attached copy of your order form, and for no other use, subject to the conditions below:



1. The Licensor warrants that it has, to the best of its knowledge, the rights to license reuse of this material. However, you should ensure that the material you are requesting is original to the Licensor and does not carry the copyright of another entity (as credited in the published version).

   If the credit line on any part of the material you have requested indicates that it was reprinted or adapted with permission from another source, then you should also seek permission from that source to reuse the material.

2. Where **print only** permission has been granted for a fee, separate permission must be obtained for any additional electronic re-use.

3. Permission granted **free of charge** for material in print is also usually granted for any electronic version of that work, provided that the material is incidental to your work as a whole and that the electronic version is essentially equivalent to, or substitutes for, the print version.

4. A licence for 'post on a website' is valid for 12 months from the licence date. This licence does not cover use of full text articles on websites.

5. Where **'reuse in a dissertation/thesis'** has been selected the following terms apply: Print rights for up to 100 copies, electronic rights for use only on a personal website or institutional repository as defined by the Sherpa guideline (www.sherpa.ac.uk/romeo/).

6. Permission granted for books and journals is granted for the lifetime of the first edition and does not apply to second and subsequent editions (except where the first edition permission was granted free of charge or for signatories to the STM Permissions Guidelines http://www.stm-assoc.org/copyright-legal-affairs/permissions/permissions-guidelines/), and does not apply for editions in other languages unless additional translation rights have been granted separately in the licence.

7. Rights for additional components such as custom editions and derivatives require additional permission and may be subject to an additional fee. Please apply to Journalpermissions@springernature.com/bookpermissions@springernature.com for these rights.

8. The Licensor's permission must be acknowledged next to the licensed material in print. In electronic form, this acknowledgement must be visible at the same time as the figures/tables/illustrations or abstract, and must be hyperlinked to the journal/book's homepage. Our required acknowledgement format is in the Appendix below.

9. Use of the material for incidental promotional use, minor editing privileges (this does not include cropping, adapting, omitting material or any other changes that affect the meaning, intention or moral rights of the author) and copies for the disabled are permitted under this licence.

10. Minor adaptations of single figures (changes of format, colour and style) do not require the Licensor's approval. However, the adaptation should be credited as shown in Appendix below.

Appendix — Acknowledgements:

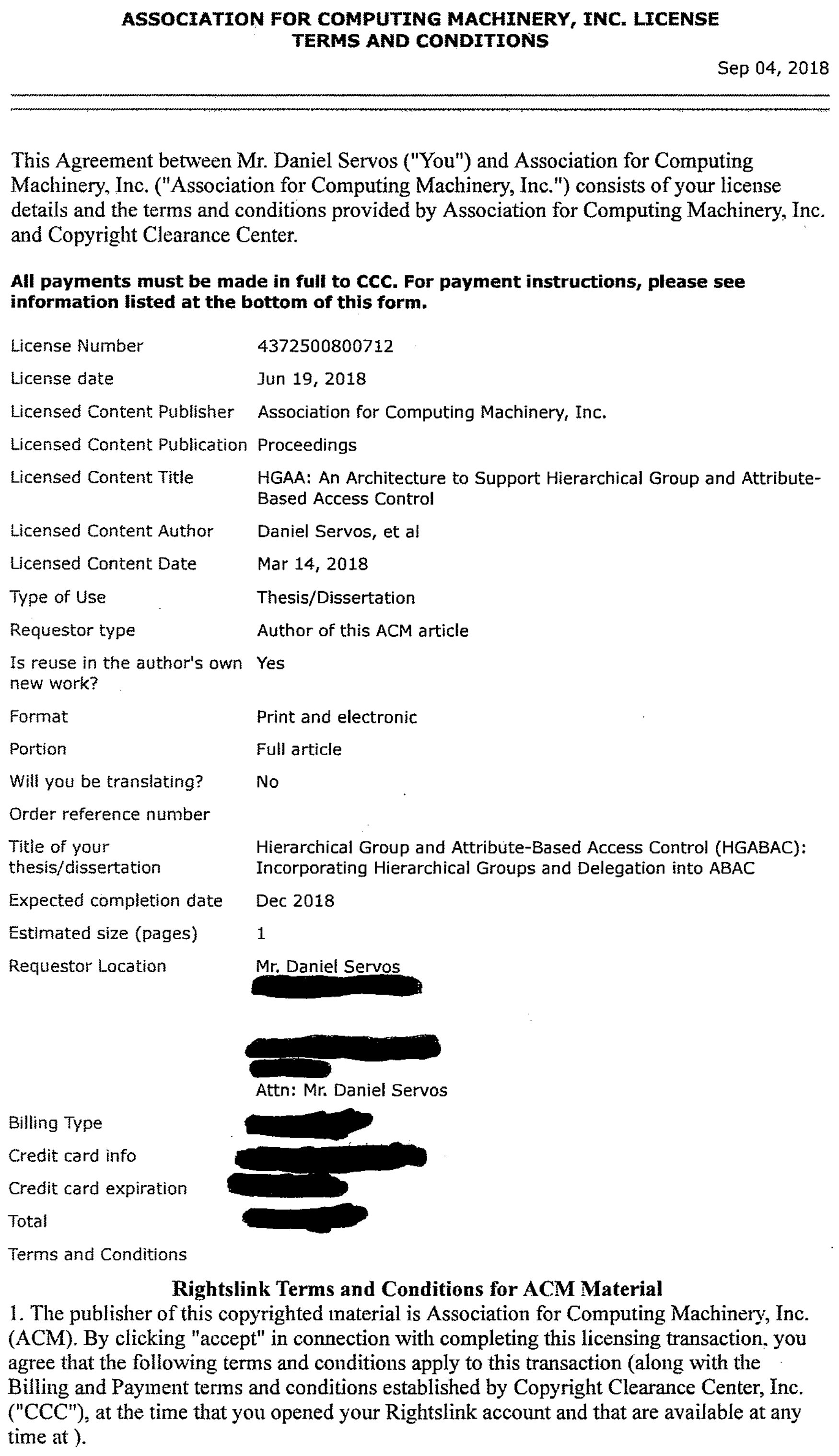
For Journal Content:
Reprinted by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication)

For **Advance Online Publication papers:**
Reprinted by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication), advance online publication, day month year (doi: 10.1038/sj.[JOURNAL ACRONYM].)

For Adaptations/Translations:
Adapted/Translated by permission from **[the Licensor]**: **[Journal Publisher** (e.g. Nature/Springer/Palgrave)] **[JOURNAL NAME] [REFERENCE CITATION** (Article name, Author(s) Name), **[COPYRIGHT]** (year of publication)

Note: For any republication from the British Journal of Cancer, the following credit line style applies:

A.4 Chapter 5: HGAA: An Architecture to Support HGABAC



ASSOCIATION FOR COMPUTING MACHINERY, INC. LICENSE TERMS AND CONDITIONS

Sep 04, 2018

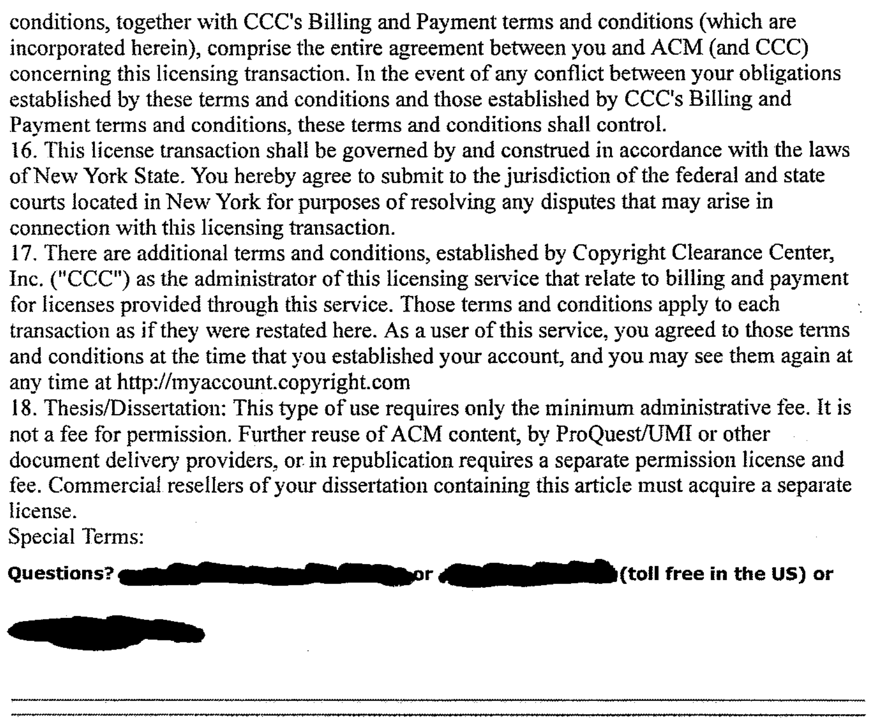
This Agreement between Mr. Daniel Servos ("You") and Association for Computing Machinery, Inc. ("Association for Computing Machinery, Inc.") consists of your license details and the terms and conditions provided by Association for Computing Machinery, Inc. and Copyright Clearance Center.

All payments must be made in full to CCC. For payment instructions, please see information listed at the bottom of this form.

License Number	4372500800712
License date	Jun 19, 2018
Licensed Content Publisher	Association for Computing Machinery, Inc.
Licensed Content Publication	Proceedings
Licensed Content Title	HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control
Licensed Content Author	Daniel Servos, et al
Licensed Content Date	Mar 14, 2018
Type of Use	Thesis/Dissertation
Requestor type	Author of this ACM article
Is reuse in the author's own new work?	Yes
Format	Print and electronic
Portion	Full article
Will you be translating?	No
Order reference number	
Title of your thesis/dissertation	Hierarchical Group and Attribute-Based Access Control (HGABAC): Incorporating Hierarchical Groups and Delegation into ABAC
Expected completion date	Dec 2018
Estimated size (pages)	1
Requestor Location	Mr. Daniel Servos [redacted] [redacted] Attn: Mr. Daniel Servos
Billing Type	[redacted]
Credit card info	[redacted]
Credit card expiration	[redacted]
Total	[redacted]
Terms and Conditions	

Rightslink Terms and Conditions for ACM Material

1. The publisher of this copyrighted material is Association for Computing Machinery, Inc. (ACM). By clicking "accept" in connection with completing this licensing transaction, you agree that the following terms and conditions apply to this transaction (along with the Billing and Payment terms and conditions established by Copyright Clearance Center, Inc. ("CCC"), at the time that you opened your Rightslink account and that are available at any time at).



Appendix B

HGPLv1 Policy Language Operations

The following table outlines the function of the comparison operations in the policy language defined in Section 3.3.2. Note that a poset element is considered to be a special case of atomic where the comparison operations $=$, $\neq$, $>$, $<$, $\geq$, $\leq$ are all based on the order of the poset but otherwise follow the above table.

Table B.1: HGPLv1 policy language operations and results.

Type	**Operation**	**Type**	**Result**
atomic	=	atomic	True if values are equal, undef if incomparable, false otherwise
atomic	=	set	True if $atomic \in set$.
set	=	atomic	True if $atomic \in set$.
set	=	set	True if sets are equal (i.e. contain same elements).
X	≠	Y	Equivalent to “NOT(X = Y)”.
atomic	IN	atomic	UNDEF
atomic	IN	set	True if $atomic \in set$.
set	IN	atomic	True if $atomic \in set$.
set$_1$	IN	set$_2$	True if $\exists s \in set_1 : s \in set_2$.
atomic	SUBSET	atomic	UNDEF
atomic	SUBSET	set	True if $atomic \in set$.
set	SUBSET	atomic	False unless $\|set\| = 1$ and $set = \{atomic\}$.
set$_1$	SUBSET	set$_2$	True if $set_1 \subseteq set_2$.
atomic$_1$	C = >, <, ≤, or ≥	atomic$_2$	True if $atomic_1\ C\ atomic_2$, undef if incomparable, false otherwise.
atomic	C = >, <, ≤, or ≥	set	True if $\exists s \in set : s\ C\ atomic$.
set	C = >, <, ≤, or ≥	atomic	True if $\exists s \in set : atomic\ C\ s$.
set$_1$	C = >, <, ≤, or ≥	set$_2$	True if $lub(set_1)\ C\ glb(set_2)$.
	NOT	X	True if X is false, false if X is true, undef if X is undef and undef if X is not a Boolean value or logical expression.
	(user.some_att)		True if $(some\_att, x) \in effective(u)$ where x is any value including NULL.

Curriculum Vitae

Daniel Servos

HTTP://CS1.CA

Education

PhD, Computer Science Candidate Sept. 2013 - Apr. 2020
Western University

- Researching the creation, standardization and formalization of Attribute-Based Access Control (ABAC) models as well as the integration of delegation, attribute groups and hierarchical concepts.
- Thesis: "*Hierarchical Group and Attribute-Based Access Control: Incorporating Hierarchical Groups and Delegation into Attribute-Based Access Control*".
- Advisors: Dr. Sylvia L. Osborn & Dr. Michael Bauer

Master of Science, Computer Science Sept. 2009 - Jun. 2012
Lakehead University

- Thesis: "*A Role and Attribute Based Encryption Approach to Privacy and Security in Cloud Based Health Services*".
- Advisor: Dr. Sabah Mohammed

Honours Bachelor of Science, Computer Science Sept. 2004 - Jun. 2009
Lakehead University

Relevant Work Experience

Lecturer Jan. 2018 - Apr. 2019
Western University

- Limited duties lecturer for CS2211b (Software Tools and Systems Programming) and CS2034b (Data Analytics: Principles and Tools) during the winter 2018 and 2019 academic term.
- Was responsible for all activities associated with the successful delivery of both courses, including but not limited to: developing, preparing, and delivering the course; setting and marking examinations, assessing the academic work of students, and reporting grades.
- Received an average teaching evaluation score of 6.4/7 (91.4%) (CS2034b) and 6.6/7 (94.3%) (CS2211b) from my students.

Graduate Teaching Assistant Sept. 2013 - Dec. 2019
Western University

- Assisted professors in the computer science department with grading undergraduate course material, tutoring students and running labs/tutorials in various computer science related courses including:
  - CS9668/4438: Internet Algorithmics
  - CS2034: Data Analytics: Principles and Tools (3 terms)
  - CS2211: Software Tools and Systems Programming (3 terms)
  - CS1033: Multimedia and Communication I
  - CS1032: Information Systems and Design
  - CS2050: The Evolution of Computing and Computers

Graduate Teaching Assistant Sept. 2009 - Apr. 2011
Lakehead University

- Assisted professors in the computer science department with grading undergraduate course material, tutoring students and running labs/tutorials in various computer science related courses including:
  - *COMP-4413: Programming Language Processors*
  - *COMP-4478: Games Design Patterns*
  - *COMP-2477: Object Oriented Programming*
  - Introductory programming courses (*COMP-0411*, *COMP-1411*, *COMP-1431*).

Research Assistant Sept. 2008 - Apr. 2009
University of Toronto

- Worked with a team on the Basie Project (an open-source classroom-friendly Python/Django based replacement for Trac).
- Developed web based modules for the project and administered the Linux based server hosting the projects web page, source control, mail list, DBMS and DNS.

Student Developer May 2008 - Aug. 2008
Google Summer of Code Program / Moodle Community

- Worked on the Moodle open-source project as part of the Google Summer of Code program to implement a plug-in for visualizing student grade and performance statistics.
- Contributed several bug fixes and patches to the project.

Thunder Bay NOW Technology Officer May 2006 - Aug. 2006
Thunder Bay Chamber of Commerce

- Planned and developed the Thunder Bay NOW website and web-based content management system as part of a CO-OP placement.

Assistant Programmer May 2005 - Sept. 2005
Canadian Water Network

- Developed web based applications for external and internal use, including applets for visualizing geographical data stored in the CWN's database.

Awards

Teaching

SOGS Graduate Student Teaching Award 2015 - 2016

- Awarded based on nominations from students and faculty. For work related to TAing CS2034 at Western University.

Research

- Ontario Graduate Scholarship (OGS) 2016 - 2017
- Ontario Graduate Scholarship (OGS) 2015 - 2016
- Ontario Graduate Scholarship (OGS) 2014 - 2015
- Ontario Graduate Scholarship (OGS) 2013 - 2014
- Ontario Graduate Scholarship in Science and Technology Sept. 2010

Teaching

Lecturing

I have taught the following courses at Western University:

- Computer Science 2034b - Data Analytics: Principles and Tools (Winter 2018 & 2019)
- Digital Humanities 2144b (Cross-listed with Computer Science 2034b)
- Computer Science - Software Tools and Systems Programming (Winter 2018 & 2019)

Refereed Publications

Incorporating Off-Line Attribute Delegation into Hierarchical Group and Attribute-Based Access Control
The 12th International Symposium on Foundations & Practice of Security Nov. 2019
Authors: Daniel Servos and Michael Bauer
Forthcoming publication in Springer Lecture Notes in Computer Science (volume number 12056)

HGAA: An Architecture to Support Hierarchical Group and Attribute-Based Access Control
Proceedings of the Third ACM Workshop on Attribute-Based Access Control Mar. 2018
Authors: Daniel Servos and Sylvia L. Osborn
DOI: 10.1145/3180457.3180459

Current Research and Open Problems in Attribute-Based Access Control
ACM Computing Surveys (CSUR) Feb. 2017
Authors: Daniel Servos and Sylvia L. Osborn
DOI: 10.1145/3007204

Strategies for Incorporating Delegation into Attribute-Based Access Control (ABAC)
The 9th International Symposium on Foundations & Practice of Security Oct. 2016
Authors: Daniel Servos and Sylvia L. Osborn
DOI: 10.1007/978-3-319-51966-1_21

HGABAC: Towards a Formal Model of Hierarchical Attribute-Based Access Control
The 7th International Symposium on Foundations & Practice of Security Nov. 2014
Authors: Daniel Servos and Sylvia L. Osborn
DOI: 10.1007/978-3-319-17040-4_12

Extensions to Ciphertext-Policy Attribute-Based Encryption to Support Distributed Environments
International Journal of Computer Applications in Technology Jun. 2013
Authors: Daniel Servos, Sabah Mohammed, Jinan Fiaidhi, Tai-hoon Kim
DOI: 10.1504/IJCAT.2013.054354

Developing a Secure Distributed OSGi Cloud Computing Infrastructure for Sharing Health Records
AIS'11 Proceedings of the Second International Conference on Autonomous and Intelligent Systems Jun. 2011
Authors: Sabah Mohammed, Daniel Servos, Jinan Fiaidhi
DOI: 10.1007/978-3-642-21538-4_24

HCX: a Distributed OSGi Based Web Interaction System for Sharing Health Records in

the Cloud
2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT) Aug. 2010
Authors: Sabah Mohammed, Daniel Servos, Jinan Fiaidhi
DOI: 10.1109/WI-IAT.2010.26

Talks and Seminars

The Roadmap to the Ecosystem of Electronic Health Records
Fifth International Conference on Digital Information Management July 5, 2010
Authors: Sabah Mohammed, Jinan Fiaidhi, David Andrew, John Thomas, Jeff Santarossa, Daniel Servos