Access Control for On-demand Provisioned Cloud Infrastructure Services

Canh Trong Ngo

Access Control for On-demand Provisioned Cloud Infrastructure Services Canh Trong Ngo

ISBN 978-94-028-0037-1

Universiteit van Amsterdam

Access Control for On-demand Provisioned Cloud Infrastructure Services

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. D.C. van den Boom
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in de Agnietenkapel
op woensdag 24 februari 2016, te 14:00 uur

door

Canh Trong Ngo

geboren te Hanoi, Vietnam

Promotor:	Prof. dr. ir.	C.T.A.M. de Laat	Universiteit van Amsterdam
Copromotor:	Dr.	Y. Demchenko	Universiteit van Amsterdam
Overige Leden:	Prof. dr.	M.T. Bubak	Universiteit van Amsterdam
	Prof. dr.	R.J. Meijer	Universiteit van Amsterdam
	Prof. dr.	P.W. Adriaans	Universiteit van Amsterdam
	Dr.	P. Grosso	Universiteit van Amsterdam
	Prof. dr.	D.W. Chadwick	University of Kent
	Dr.	L.H.M Gommans	KLM

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 346.
This research was supported by FP7 EU funded projects The Generalised Architecture for Dynamic Infrastructure Services (GEYSERS, FP7-ICT-248657).

Typeset by LaTeX.
Printed and bound by Ipskamp Drukkers
ISBN: 978-94-028-0037-1

To my family...

Contents

Chapter 1

Introduction

Cloud Computing is effectively used to improve scalability, availability, elasticity and security of IT management in many application areas. It adopts advantages of many technologies such as virtualization, service-oriented architecture, Grid Computing and Utility Computing to allow customers and providers to cut costs on system deployments and operations. Many studies and best practices documents related to clouds deployment, design, development, operations and management have been proposed to incorporate above technologies [1–6]. Clouds in such approaches enable users' data to store on share virtualized cloud infrastructures, which are on-demand provisioned at providers' facilities. The virtualized infrastructures capacities can be elastically scaled up and down depending on varying users' demands. In clouds, the diversity of accesses on shared resources brings challenges to protect confidentiality, integrity and availability. The cloud systems must guarantee unauthorized parties cannot access or modify protected resources. Therefore, access control is one of the crucial component in the cloud security. In this thesis, we focus on designing flexible and efficient access control mechanisms to protect cloud resources and simultaneously inter-operate with cloud infrastructures of providers.

1.1 Cloud Computing Characteristics

Cloud Computing was presented as the prospective development model of distributed computing which would influence the whole IT industry. It was initially introduced as a new product on utility computing by Amazon Web Services (AWS) [7], along with different concepts and definitions [1, 3, 5, 8–10]. It has been developed to address scales of data and service processing in both scientific and industry communities with efforts to reduce consumers' costs and optimizing resource utilization. Although there are many Cloud Computing definitions [1, 3, 5, 8], the most popular one is from US. National Institute of Standards and Technology (NIST) [5] including essential characteristics, deployment models and service models. Figure 1.1 presents the conceptual reference model of the NIST Cloud Computing architecture.

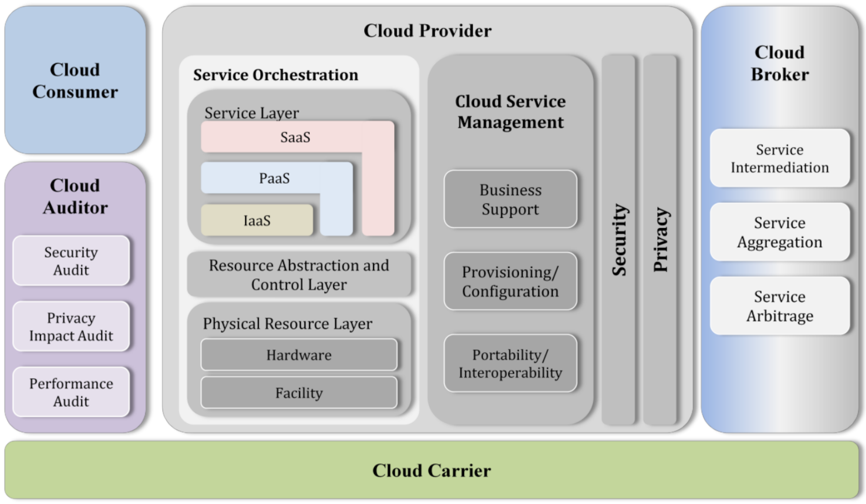


Figure 1.1: The NIST Cloud Computing conceptual reference model [9]

According to NIST [5], Cloud Computing has the following essential characteristics:

- On-demand self-service: consumers can obtain services as and when needed without requiring human interaction.
- Broad network access: Users can access services using wide range of network clients such as mobile phones, tablets, laptops, and workstations.
- Resource pooling: the provider's computing resources are not dedicated to particular consumers but are pooled to serve multiple consumers using a multi-tenant model.
- Rapid elasticity: the amount of allocated resources can be provisioned and expanded rapidly commensurate with consumers' demands.
- Measured service: amount of resources consumed is measured by a metering capability that appropriates to the type of services, (e.g. storage, processing, bandwidth, and active accounts).

NIST also defines other Cloud Computing aspects such as service types, deployment models, use-cases and business opportunities [5]. The classification of cloud services relates to the separation of responsibilities between providers and customers on system managements. Prior to the clouds, customers were responsible for their whole computing system stack, from hardware devices, operating systems to middleware, and applications. Normally they were all deployed on-premise at customers' sites. Cloud Computing was then proposed to provide separations of responsibilities on layers, in which the lowers were essentially outsourced to

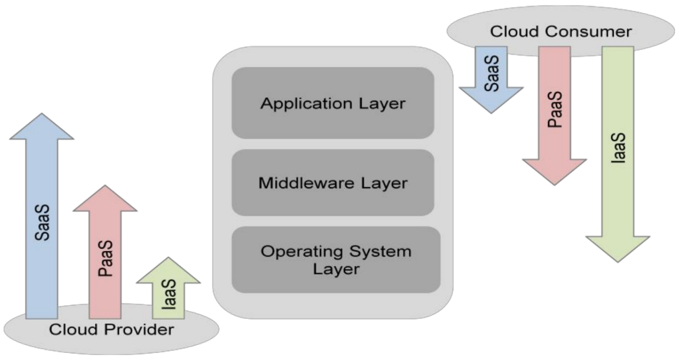


Figure 1.2: Scope of controls between provider and consumer in NIST cloud services [9]

providers. It also provided the scalability and cost reduction in deployment, operation and management. Depending on either cloud providers or customers are in charge of what parts, cloud services can be classified into the following types:

- Infrastructure as a Service (IaaS): providers take care the networking, storage, computing and virtualization platforms, while customers control operating systems, middleware, runtime libraries and applications. Examples of this cloud service type are the Amazon EC2 [11], Microsoft Azure Compute & Network Services [12], Google Compute Engine [13], and Rackspace Managed cloud [14].

- Platform as a Service (PaaS): In this model, operating systems, middleware and runtime libraries are managed by providers. The customers need to focus only on their deployed applications and data. Typical PaaS platforms are Google App Engine [15], Microsoft Azure [12], Salesforce Heroku [16].

- Software as a Service (SaaS): this type of service gives customers the whole application service as if it is running on-premise. The difference is that providers take care on management of all the system from hardware, operating systems, networking to application deployment. This service model can be seen in Google Apps [17], Apple iCloud [18] and Microsoft Office 365 [19].

The Figure 1.2 illustrates the separations of controls between providers and consumers in mentioned cloud service types.

These cloud services can be deployed in different deployment models [5]: (i) private cloud, (ii) community cloud, (iii) public cloud or (iv) hybrid cloud. Actors in clouds use-cases are therefore defined according to the respective cloud architecture [9], including cloud consumer, cloud provider, cloud auditor, cloud broker and cloud carrier.

1.2 Convergence of Cloud Infrastructures and Optical Network Virtualization

In the research efforts for developing advanced cloud infrastructures using future network technologies, GEYSERS project [10] developed concepts and solutions corresponding to the above Cloud Computing characteristics.

- Huge increase in the number of users/applications and a rapid increase in available bandwidth for users beyond 1Gbps: GEYSERS defined and developed a novel dynamic wavelength service provisioning mechanism that enables network operators to manage their capacity to support large number of users with high bandwidth optical connectivity.
- High bandwidth requirement applications with 10Gbps or more become more popular in transferring data between data centers, networks for HD and SHD multimedia content distribution, large remote sensor networks or huge scientific data. GEYSERS defined and implemented a new mechanism for network operators to request and setup scheduled high bandwidth optical network connectivity between endpoints in an on-demand manner.
- GEYSERS provided multi-domain, inter-provider network and IT resources to users. They are managed by the consistent, dynamically provisioned security and access control policies.
- For applications requiring large-scale convergence of IT and network services which are similar to Amazon virtualized services and Microsoft SharePoint, GEYSERS contained a novel end-to-end service provisioning mechanism that automatically and efficiently bundles suitable IT resources with the required optical network connectivity services to provide to the user in a single step in an on-demand manner.
- GEYSERS defined and developed methods allowing infrastructure providers to partition their resources (optical network or IT resources) to compose logical infrastructures and offer to network operators as a service. The logical composition mechanism supported dynamic and on-demand changes of combined optical network and IT resources.

The GEYSERS reference model is illustrated in Figure 1.3.

The project was aiming to provide the coordination between optical networks and IT resources across multiple provider domains, including infrastructure providers and network operators. Based on the novel mechanism to partition infrastructure resources to compose and deployed at different providers, the architecture implies a new business framework with cost and energy efficiencies. Such features bring requirements on architecture and system design as well as implementation [20].

GEYSERS models entities participating to specific workflows defined in its use-cases [21]. Nowadays, telecom and IT service operators use integrated roles

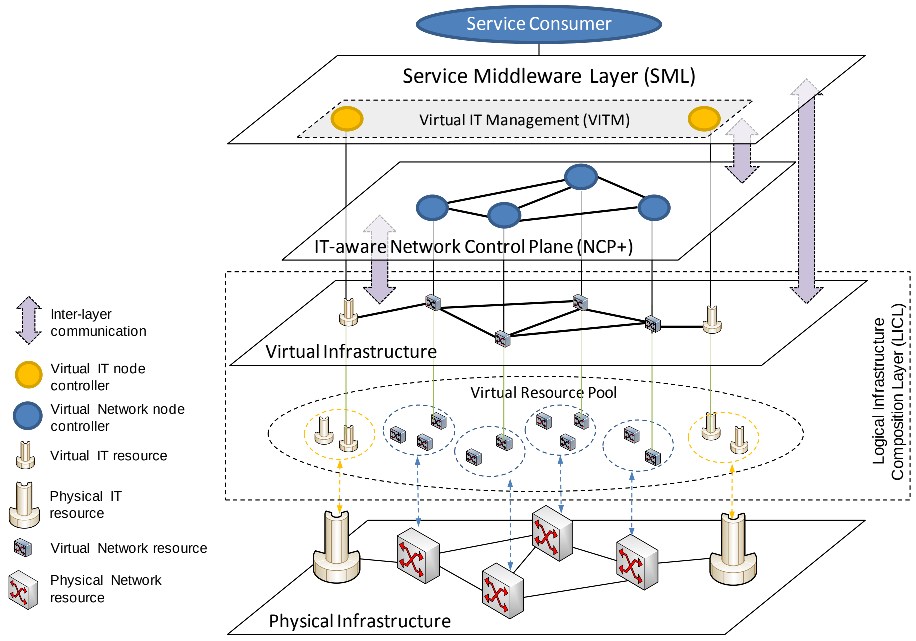

Service Consumer
Service Middleware Layer (SML)
Virtual IT Management (VITM)
IT-aware Network Control Plane (NCP+)
Virtual Infrastructure
Virtual Resource Pool
Logical Infrastructure Composition Layer (LICL)
Physical Infrastructure
Inter-layer communication
Virtual IT node controller
Virtual Network node controller
Virtual IT resource
Physical IT resource
Virtual Network resource
Physical Network resource

Figure 1.3: GEYSERS reference model

of infrastructure providers and infrastructure operators. It is expected they own the physical infrastructure, operate it, and finally, run services on top of it. This approach is highly inflexible, inefficient and extremely expensive as a whole, but not in specific business sub-processes. In the new architecture, GEYSERS enhances the current business models with following roles to reflect the importance of virtualization of the network and IT infrastructure:

- Physical Infrastructure Provider (PIP): The PIP role in GEYSERS implements the possession and operation of the physical infrastructure.
- Virtual Infrastructure Provider (VIP): The VIP role implements the virtual resource handling and composition for producing Virtual Infrastructures (VIs).
- Virtual Infrastructure Operator (VIO): The VIO role implements the configuration and operation of the infrastructure and also provides final services to consumers.

The research presented in this thesis is carried out in the context of and supported by the GEYSERS project [10]. The proposed architecture encounters challenges on designing and implementation access control mechanisms for distributed, inter-domain, multi-provider environments. It motivated us to perform research on access control approaches for inter-domain and multi-provider cloud infrastructure systems. The following section defines necessary requirements for access control in cloud infrastructure systems.

1.3 Access Control Requirements for Clouds Service Providers

From Cloud Computing characteristics identified by NIST and in the context of multi-provider cloud infrastructures in GEYSERS, the access control for cloud infrastructure systems should contain following features:

- **On-demand provisioning and self-configuration**: cloud resources are normally allocated and adjusted dynamically according to customers' requirements. They are reflected by the on-demand updating of resource meta-data. The access control mechanisms to manage cloud resources should bind and synchronize to these dynamic meta-data.
- **Fine-grained access control**: The access control mechanism must support rule-based definitions which are flexible enough to adapt different access control use-cases. More specific, it should provide ways to care about rich conditions requirements (i.e., *who, what, when, where, why* and *how* clauses in authorization statements) to consent accesses.
- **Flexible multi-tenancy**: An essential characteristic of clouds is to serve pooled resources for multiple customers following the multi-tenant model. In this manner, the proposed access control system must provide the multi-tenancy features by design.
- **Scalability**: A cloud provider should be able to manage its virtualized resources to serve large scale of customers. It requires that system performance must be critical in adopting suitable access control mechanisms.
- **Distributed**: Business models and use-cases of cloud infrastructures target for distributed environments. Therefore we need to provide mechanisms to inter-operate access control systems among multiple distributed cloud providers.

In the next section, we revisit existing access control models and most popular policy languages to identify what can be used in the cloud security. They will be used to justify our research goals in section 1.5.

1.4 Related Work

1.4.1 Preliminaries on Access Control Models

Access control has the purpose to protect the confidentiality and integrity of the system information. Besides authentication systems are in charge of confirming the truth of users identities, the access control regulates subjects' operations on data and resources. Based on functionalities, an access control system can be separated into the decision and the enforcement components [22, 23]. In the Access Control Framework for Open Systems ISO 10181-3 [22] illustrated in Figure 1.4, the initiator is the active subject who submits access requests to the Access

Control Enforcement Function (AEF). Here they are mediated by sending decision requests to a Access Control Decision Function (ADF) which determines whether they should be granted or denied. Depending on responded decisions, the AEF will enforce requests, e.g. either subjects can access the resources or denied messages are thrown.

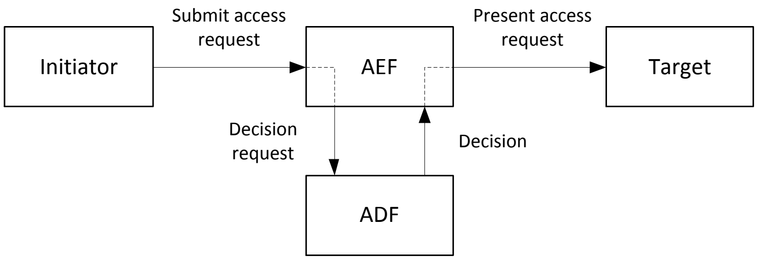


Figure 1.4: ISO 10181-3 access control framework [22]

In [23], Policy Enforcement Points (PEPs) at resource locations send decision requests to the Policy Decision Point (PDP), which is in charge of giving decisions from the predefined policies. In this way, policies can be stored and managed separately from resources. Access control models therefore are proposed to define interactions between PDPs and PEPs, as well as manage policies in authorization systems. Although there are many different access control models for various systems, they can be classified into following types:

1.4.1.1 Discretionary Access Control

In the Discretionary Access Control (DAC), a resource is assigned the ownership to one or more entities. The owners have all controls to decide who can access the protected resources and which permissions they allow to do. Policies in DAC models are implemented by either the access control matrix or the access control list [24, 25].

1.4.1.2 Mandatory Access Control

The Mandatory Access Control (MAC) is designed to prevent illegitimate information leakage based on clearance definitions. The data owners cannot set permissions like in the DAC. Instead of that, resources and subjects are attached security labels. The system configuration defines access rules based on labels and enforce them strictly.

The typical access rules in MAC are security clearance levels: i.e., labels could be in partial order-sets: top-secret, secret, confidential, restricted, official, unclassified, clearance [25]; or in categories, in which different areas are disjoint and competence (e.g. the Chinese Wall model [26]). The best known model to secure information flows is Bell-LaPadula model [27] which uses both MAC and DAC.

	On-demand self-service	Fine-grained authorization	Multi-tenancy	Scalability	Distributed
DAC	-	-	-	-	-
MAC	-	-	-	-	-
RBAC families	-	-	✓	-	✓
ABAC	-	✓	-	-	✓

Table 1.1: Access control models comparisons

1.4.1.3 Role-Based Access Control

The Role-based Access Control (RBAC) families [28–30] are introduced with roles as an abstraction layer decoupling users and permissions. Rather than assigning directly to users in DAC models, the basic $RBAC_0$ model groups permissions into roles according to task descriptions, which is known as the role engineering process. Users assigned to roles will contain all permissions of the active roles. In extensions, roles can be organized in hierarchy in $RBAC_1$ [29], constraints to limit user and role assignments with $RBAC_2$ and $RBAC_3$. In practical, RBAC approaches are applied in different databases and operating systems. The common feature of these systems is that they have stable structures in which tasks are defined static. Hence, role organizing after the role engineering process mostly is unchanged. Rather, RBAC systems have drawbacks to support fine-grained authorizations such as dynamical assigning roles according to variable contexts (e.g. time, location, authorization states), which may lead to role explosion problems [31, 32].

Some extended RBAC approaches are proposed to support multi-tenancy features for clouds [33–35]. However they either have drawbacks in dynamic on-demand supports, fine-grained authorization and scalability [33, 34] or lack of practical mechanisms [35].

1.4.1.4 Attribute-based Access Control

To overcome limitations of RBAC systems, Attribute-based Access Control (ABAC) model is identified with the central idea that access can be determined based on present attributes of objects, actions, subjects and environment in the authorization context. It means that the ABAC is more flexible and scalable for real-time environments than RBAC. However, managing large number of attributes in ABAC is a challenge. Also, the complexity of attributes criteria in rules and conflict resolutions may arise during applying ABAC in access control for large-scale systems like cloud. ABAC implementation like eXtensible Access Control Markup Language (XACML) [36] only defines a general ABAC policy language with fine-grained authorization capability. Making it applicable with features analyzed in Section 1.3 requires both extensions and practical implementation mechanisms. This is the purpose of this thesis.

The Table 1.1 summarizes access control models adapting requirements in section 1.3.

In ABAC, the flexible and fine-grained authorization capabilities depend on the expressiveness of the provided policy language. Mechanisms such as AWS Identity and Access Management (IAM) JSON-style authorization policy [37], XACML standards [36, 38] provide different flexible degrees. For simplicity purposes, Amazon JSON-style policy [37] is quite limited compared to XACML [36] such as policy hierarchy, combining algorithms, attribute comparisons, etc. In this thesis, we adopt XACML [36] to present our authorization rules and policies. However policies with the high expressiveness also have their complexity. So to apply XACML successfully, we need to investigate approaches in our implementation. Next section gives a brief overview of XACML to identify its implementation challenges.

1.4.2 Access Control Policy Languages

1.4.2.1 Ponder

Ponder [39] is a declarative, object-oriented policy language designed for distributed object systems. It consists of five types of policies: the authorization policies could define positive or negative decisions upon matching of subject, target resource, action and the optional environment constraint; the filtering policies extend by associating an action with filter expressions, where input or output parameters are defined; the delegation policies are used to transfer access rights defined in an authorization policy temporarily; refrain policies bind to subjects to avoid operations in spite of permitted by authorization policies; finally, the obligation policies define tasks must be performed upon specific events occurs. Ponder provides the ability to organize policies in groups and roles in hierarchy to reflect organizational structures, so supporting RBAC features.

However, Ponder has some limitations when applying in Cloud Computing. First, the language does not mention how to process and evaluate policies. The Ponder Toolkit only contains a compiler to transform policies in to a system dependent language (e.g. Java code). Secondly, in distributed environment, policies can be defined by different authorities while Ponder does not define policy issuers, thus assumes that all policies are trusted. The policy repository must implement an enforcement mechanism to manage trusted policies. And finally, the Ponder implementation is obsoleted and no longer supported [40].

1.4.2.2 PERMIS

PrivilEge and Role Management Infrastructure Standards (PERMIS) [41] is a role-based access control infrastructure using X.509 attribute certificates [42] to store users' role values. In this system, their integrity is protected by digital signatures of Attribute Authorities (AAs). Storing attributes in X.509 certificates is similar to the Security Assertion Markup Language (SAML) [43] with the XML-based structure. To implement RBAC concepts, PERMIS uses permission attributes and role attributes. RBAC operations are defined by a XML-based policy language containing: subject domain policy to define group of subjects, role hierarchy policy to define role hierarchical relationships, policy to define assignments from roles to a subject domain with related validity periods and delegation depths (a.k.a the role assignment in RBAC), action policy to specify actions binding with a target resource, and policy to assign actions to a role under conditions (a.k.a the RBAC permission assignment). Policies are created by Privilege Allocators at AAs, then are stored in Lightweight Directory Access Protocol (LDAP) directory services. The system follows the ISO 10181-3 Access Control Framework [22]. In this framework, the ADF retrieves policies from LDAP services, along with attribute certificate revocation lists to validate principals' attribute certificates.

In general, the XML-based policy language in PERMIS tried to implement RBAC operations with X.509 attribute certificates. Therefore, it suffered drawbacks from RBAC models with the stable role structure. The policy to assign actions to role containing conditions with Boolean logic could provide some levels of fine-grained authorization. However, policies in PERMIS have only two decisions: either permit or deny-by-default, so it cannot handle well intermediate results upon errors or undefined cases (i.e., conflicting decisions or undefined policies) as in the XACML.

1.4.2.3 XACML

XACML [36, 38] is an authorization policy language in XML format based on the ABAC model. It composes policies from set of attribute criteria joined by logical operators, which are used to decide answers for authorization requests. XACML is scalable in arrange policies in hierarchy order which can be combined and extended vertically by conflict resolution algorithms. In addition, by supporting delegations, obligations and advices, XACML is applicable in many areas such as networking, grids, clouds, enterprise organization and management. However, the growth of policies to address system scales will increase complexity of the policy repository, which leads to the drop of policies evaluation performance.

XACML v3.0 [36] organizes components following the model in Figure 1.5. In the model, policies are arranged hierarchically as follows:

- A policy-set is composed from a set of policies or other policy-sets.
- A policy can contain a set of rules.
- A rule has a logical expression of attribute values as the criteria for rule's decision.
- Parent's decisions of policy elements are the joined of children's according to the predefined combining algorithms.

The XACML is considered as a *de facto* standard for the attribute-based authorization policy language. Adopting XACML allows systems to have advantages supporting complex fine-grained authorization scenarios than other policy language. In spite of that, due to complexities in its evaluation semantics, there are gaps to apply XACML standards for high performance systems.

The dynamic and elastic features in Cloud Computing identified in section 1.1 require a flexible access control model like the ABAC for multi-tenant systems. It requires a rich expressive policy language supporting sophisticated, fine-grained authorization rules. However, the implementation mechanism needs to have efficient and scalable performance to adapt large-scale cloud management systems at providers. In the next section, we identify and formulate research questions on how to resolve these challenges.

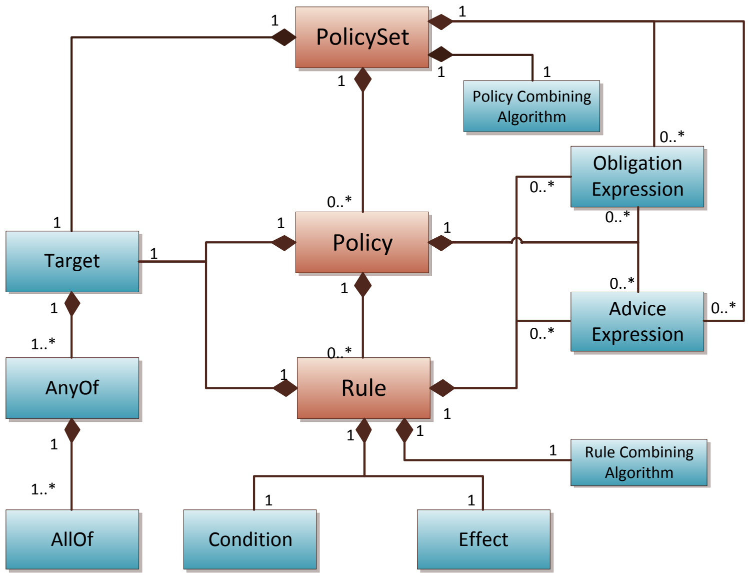


Figure 1.5: XACML 3.0 policy model [36]

1.5 Research Questions

In this thesis, we focus on model designs and implementation techniques for access control solutions for cloud infrastructure services. They are investigated by the following research questions:

1. *How do we design a flexible and scalable access control model supporting the on-demand provisioned self-service of cloud infrastructure services?*

In order to support large numbers of users with elastic resource scaling, cloud providers need to apply on-demand provisioning mechanisms to manage their resource capacity. As depicted in section 1.2, they are either the optical wavelength service provisioning for network operators, or the computing and storage share resource pool mechanism for IT service providers. In general, these mechanisms generate changes in numbers and properties of resources over fine-grained time resolutions, e.g. minutes or hours. However, existing access control models are eventually designed to manage a stable numbers of resources. Whenever there's a change in the system e.g. adding new or removing old resources, policies should be reconfigured, such as role engineerings in RBAC or updating attribute-based policies in ABAC. There is no access control model that is aware of life cycles of provisioned resources. We need a formal model along with its mechanisms binding between service life-cycles of cloud infrastructures and their access control policy management. This question will be investigated and answered in Chapter 2.

2. *How do we design an access control model for cloud infrastructure providers*

in which customers can manage their own virtualized resources distributing in multiple domains?

From cloud providers' perspective, cloud systems should support sharing resource pools to serve multiple customers, where they are able to customize and exploit subscribed services in an isolated manner. It is known as the multi-tenancy [5, 44]. In this manner, the access control model for clouds providers should be designed with multi-tenant features. Tenants are able to customize policies for their services, while still under the scope of the provider's policies. The NIST cloud definitions [5] said that cloud services are able to scale up extensively, not only from a single provider, but also from multiple, distributed providers. It is illustrated in the GEYSERS approach [10], in which partitioned resources at multiple providers can be aggregated to compose customers virtual infrastructures [21]. Therefore cloud providers should collaborate to offer inter-provider cloud infrastructures composed from resources crossing distributed domains. Such sophisticated systems are usually provisioned in multiple domains using service lifecycle managements [45, 46]. Designing the access control model for inter-provider systems should be aware of these characteristics. We solve this question in chapters 2 and 3.

3. *How do we implement a high performance authorization policy evaluation engine, which should be required in access control solutions for cloud providers?*

Mechanisms for access control models provide different expressiveness. They represent how much flexibility policies can be configured to moderate managed resources, i.e., defining criteria in clauses *who*, *what*, *where*, *when*, *how* of authorization statements. However, access control is eventually one of overheads of systems. Such mechanisms should be aware of the overall system performance of cloud providers, which should handle hundreds or thousands of customers at the same time. Normally, the more expressive the policy language is, the more overhead the access control adds to the system. In this thesis, we adopt XACML as the attribute-based policy language for fine-grained authorization purposes. The overhead when applying XACML motivates us to investigate and introduce the mechanism in chapters 4 and 5 to efficiently improve system performance.

1.6 Contributions

The contributions in this thesis are as follows:

- Chapter 2: We propose an access control model for multi-tenant cloud services using attribute-based policies. The model could integrate with the existing information model of cloud resource managements, which allows dynamic updates and reconfigurations regarding system scaling. It not only has scalability in terms of number of resources, but also allows delegations and collaborations among tenants in multiple levels.

- Chapter 3: In this section, we extend the proposed multi-tenant access control model for distributed multi-domain Intercloud infrastructures. It used the proposed token exchange mechanisms among providers aiming to guarantee model's characteristics in the distributed environment. The extension is then validated in set of Intercloud scenarios involving multiple cloud providers.

- Chapter 4: XACML is a widely used attribute-based policy language that contains rich functionalities and expressiveness. To facilitate the adoption of this policy language in our access control model, we analyze and express the XACML logical model, then design new decision diagram data structures to represent its elements. These mechanisms could be applied to solve problems in different XACML policy managements, including the high performance policy evaluation engine in the next chapter.

- Chapter 5: using defined mechanisms in the previous section, we build up a high performance XACML policy evaluation engine. It not only has magnitudes of throughputs compared to previous work, but also maintains the same policy semantics and expressiveness [47]. The engine is used as the basis for building high performance PDPs in our multi-tenant access control model which is implemented and deployed for complex cloud infrastructure services.

Finally, Chapter 6 summarizes results presented in previous sections to answer the identified research questions.

Chapter 2

Multi-tenant Access Control for Single Cloud Providers

This chapter is based on the following publications:

- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Security framework for virtualised infrastructure services provisioned on-demand," in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 698–704 [48].
- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures," in Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 343–349 [49].
- C. Ngo, Y. Demchenko, and C. de Laat, “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services," in Journal of Information Security and Applications (*accepted 2015*) [50].

2.1 Introduction

Cloud Computing is emerging as a common service model approach for on-demand infrastructure services provisioning, including computing, storage and networking. It allows minimizing infrastructure management costs for both customers and providers. Besides a wide spectrum of currently available cloud services, there are numbers of research and standardization activities focusing on definitions, use-cases, and reference models such as NIST clouds [5, 6], OGF ISOD-RG [51], and OASIS Cloud Identity [52].

Current cloud concepts identify five essential characteristics [5]: (i) on-demand self-service; (ii) broad network access and diversity of client devices; (iii) resource pooling that allows to serve multiple customers using a multi-tenant model, by managing resource utilization more efficiently with virtualization, partitioning and workload balancing; (iv) rapid elasticity that allows scaling resources dynamically;

(v) measured service with the pay-per-use business model. Other additional feature is the heterogeneity on both provider and customer sides, and multi-provider services.

With all these characteristics, providing consistent security and privacy solutions for Cloud Computing environment brings many challenges [53]. Several surveys and researches have shown that the security and privacy are the main obstacles to widely adopt Cloud Computing [6, 54, 55].

Based on Cloud and Intercloud scenarios analyses from GEYSERS [10] and GEANT3 [56] projects, this chapter presents an access control model for multi-tenant cloud services using attribute-based policies. The extended model is applied for Intercloud scenarios with the exchanging tokens approach for fine-grained dynamic trust establishment. To facilitate attribute-based policy evaluation and implementing the proposed model, we apply a new mechanism to transform complex logical expressions in policies to compact decision diagrams. Our prototype of the multi-tenant access control system for Intercloud is developed, tested and integrated into the GEYSERS project. Evaluations demonstrate that our system has good performance in terms of number of cloud resources, clients and policies.

2.2 Related Work

A number of approaches and contributions in access control for cloud service management have been proposed with the emergence of Cloud Computing in 2010-2014. Based on the Common Information Model (CIM), authors in [33, 34] proposed an RBAC model integrating with the CIM. In this work, an authorization statement is defined as the 4-tuple of ⟨*issuer, subject, privilege, resource*⟩ written as a rule using Semantic Web Rule Language (SWRL) [57]. The rule is then reasoned by a DL Reasoner to transform into statements using Resource Description Framework (RDF). Work at [34] illustrated it is possible to use proposed model to support RBAC features for users of a tenant. Inter-tenant collaborations are represented by sharing context information, so the trustee can define authorization statements. However, they do not support multiple level delegations among tenants as well as granularity of inter-tenant trusts is limited. Besides that, if cloud systems scale up with number of resources, subjects (tenants and users), complexities of policies with number of authorization statements, the DL reasoner mechanism could be the potential bottleneck when number of RDF statements may explode. Moreover, by utilizing SWRL, an OWL DL implementation, as the access control language, the expressiveness and flexibility of the policy language is limited compared to other policy languages like XACML.

The Multi-tenant Role-based Access Control (MT-RBAC) [35] extended the basic RBAC with a set of models including administration features. Beside regular intra-tenant permission and role assignment operations, the cross-tenant collaborations are performed by sharing roles. The trustor tenant can define either all roles to trustee tenants (MT-RBAC$_0$), the same public roles to all trustees (MT-RBAC$_1$), or separated public roles to different trustees (MT-RBAC$_2$). In turn, the trustee can perform two administrative operations: (i) user assignment (UA) to its users and

(ii) role hierarchy on the shared roles. The prototype was carried out using the attribute-based policy language XACML with the RBAC profile extension.

Authors in [58] extended the ABAC model from [59] to IaaS scenarios. In their model, entities were classified into cloud root user who can manage VI and tenants; tenant root user who can configure attribute profile and manage tenant admin users; tenant admin users in a tenant can manage tenant regular users and finally tenant regular users who can operate on cloud resources. Using the policy language defined in [59], the prototype was integrated with an OpenStack system. Although the approach used ABAC applying for multi-tenant scenarios, its model did not aware of policy conflict problems in multi-tenancy when multiple entities can define policies. Thus, they also did not contain isolation and grant constraints for policies as in our approach, which would resolve the policy conflict problems. In addition, their model was not aware of collaborations between tenants.

To protect data in outsourced environments like clouds, Attribute-based Encryption (ABE) research [60–62] was proposed for security of outsourcing storage, while homomorphic encryption [63, 64] was proposed to secure computation on hostile systems. In the key-policy ABE approach (KP-ABE) [61], request attributes were associated to ciphertexts, and policies were associated to users' keys. The ciphertext-policy ABE (CP-ABE) scheme [62] provided a mechanism that allows creating users' keys based on their attributes, and attribute-based policies to protect data are associated in ciphertexts. The ABE schemes were extended and applied to secure data on cloud storage services [65, 66]. Although the homomorphic encryption may provide confidentiality in outsourcing computation, its applications on cloud were still limited due to the complexity and performance overhead [67]. All these cryptographic mechanisms can be seen as the AEF in the ISO 10181-3 access control framework [22], while our work focuses on access control decision components. Therefore, ABE and homomorphic encryption are orthogonal with our proposal.

In collaboration environments, delegation of rights was investigated to support dynamic decisions in single or multiple security domains [68–71]. Related works on delegation were analyzed in detail in [68] which provided comprehensive taxonomy of delegation methods and mechanisms. Delegation models using exchange tokens were popular in previous work [71–74]. Authors in [70, 71] proposed a general model for dynamic delegation of authority for multi-domain authorization that used the Credential Validation Service to validate subject credentials across multiple security domains. In our work, we use delegation types concepts adopted from [69] and an authorization token based mechanism to transfer security contexts between domains. In addition, the proposal in section 3.3 solves the constraints synchronization issue in on-demand provisioning cloud infrastructure, which was addressed in previous works.

AWS IAM [37] is the integration of an identity management system and an access control mechanism. Upon subscribing to an AWS product, each customer is assigned an AWS tenant account. All operations on AWS products are then bound to this account. AWS IAM provides a mechanism to create and manage multiple users binding to the AWS tenant account. Using JSON-style authorization policies storing at the IAM side or attaching at the AWS product side, the IAM could control user activities on AWS resources. To guarantee security requirements on confidentiality and integrity, users are allocated their own security credentials to access AWS resources. However, supporting policy language in AWS IAM is not very expressive with simple attribute-based policies with limited RBAC features. Cross-account access is defined by creating an IAM policy of the trustor account to the trustee account. The trustee then can delegate these privileges to its users. It does not support multiple-level cross-account collaborations.

OAuth authorization framework [74] enables a third-party to access a HTTP resource by approval of the data owner via tokens. It provides a workflow protocol for distributed authorization currently applied in various cloud-based services such as Google APIs and Twitter APIs. However, OAuth authorization framework only stops at defining a distributed authorization workflow for HTTP resource and does not specify authorization policy definitions.

Compared to the related work, our proposal resolves the access control problems for multi-tenancy more complete. We formalize the ABAC in the multi-tenant model with isolation and grant constraints, that prevent policy conflict problems. Our approach also supports flexible collaborations between tenants with multiple levels of delegation. The extended model for Intercloud in Chapter 3 uses the token exchange approach which synchronizes constraints and applies cryptographic mechanisms to solve the distributed authorization with delegation constraints issues, which is more suitable for our approach than the current OAuth 2.0 framework [74].

2.3 Problem Statement

A cloud platform should support multi-tenant design that is capable of sharing services for different tenants as if all of them are using dedicated systems [44, 75]. In this sense, the access control services provided for tenants to manage their cloud resources must support multi-tenancy features, including isolation (e.g., performance, administration); on-demand customization of access control configuration such as authorization policies, identity management, trust anchors; and bindings between cloud systems and tenant on-site services (e.g., access control for clouds could interconnect to an on-site LDAP directory service of the tenant).

The on-demand self-service and rapid elasticity properties [5] in clouds require that the access control design must handle dynamic changes of entities in authorization scenarios. For example, a typical cloud IaaS service provides different plans (e.g., storage size, speed, computing powers, bandwidth, lifetime, etc) for the number of subscribed customers may reach thousands. Each of them could then manage hundreds of end-users. In such cases, numerous resource objects are provisioned over time with dynamic identifiers. Thus, the cloud management platform must handle accesses from users using diversity of clients in both types and numbers (e.g., mobile devices, laptops, workstations) to access these resources. Moreover, the access control for cloud services should also support rich context attributes (e.g., time, location and types of clients) for the fine-grained authorization. Such challenges need to be solved in a dynamic robust access control approach for cloud services.

Traditional access control models are designed to manage accesses from subjects to objects with specific operations via authorization statements. A trivial statement is a triple of $\langle subject, object, operation\rangle$, in which the $\langle object, operation\rangle$ is known as a permission. RBAC approaches [28–30] were introduced with roles as an abstraction layer decoupling subjects and permissions. RBAC was supported to apply in different areas, from stand-alone, enterprise-level or cross-enterprise applications. However, even the design purpose of RBAC is to large enterprise systems with even hundreds or thousands of roles and users in tens thousands [76], such systems may have problems on scalability in role and object explosions [31, 32]. Analysis in [32] estimates that RBAC should be used for systems with static structure where roles and hierarchy are clearly defined; entities individuality and locality are limited; and managed objects are stable. However, large-scale cloud services management systems often have dynamics of provisioned pooling objects, varieties of entities and sophisticated fine-grained authorization regarding dynamical context-specific attributes, in which RBAC approaches may not be suitable.

To overcome limitations of RBAC systems, ABAC was identified with the central idea that access can be determined based on present attributes of objects, actions, subjects and environment in the authorization context [59, 77, 78]. The ABAC can be used to model RBAC as well as other traditional access control models [59]. The fine-grained authorization feature of ABAC makes it more flexible and scalable than RBAC. Thus, ABAC is mostly suitable for cloud management services. For example, a cloud provider may allow different accesses from users of a customer A to a set of subscribed cloud resources (e.g., storage, Virtual Machine (VM), database) during subscribing time. In turn, part of these resources are shared read-only to others, e.g., external consultants from an auditing firm can read during working-time in a month.

However, using large numbers of attributes in ABAC produce challenges in management and deployment. The complexity of attributes criteria in rules and conflict resolutions may arise during applying ABAC in access control for large-scale systems like cloud. ABAC implementation like XACML standard [36] only limits at defining a general ABAC policy language but without indicating how to integrate with system resource information models for attribute management, as well as defining necessary constraints in policy composition and management for multiple authorities like the multi-tenant systems.

With all such challenges and motivated by cloud and Intercloud scenarios analyses [10, 48, 49, 56], as well as related work on access control for clouds [33–35, 37, 58, 60–62], we introduce the Multi-tenant Attribute-based Access Control (MT-ABAC) approach which formalize the ABAC applied for the multi-tenancy pattern. It not only aims to provide a scalable and flexible resources and entities management of the ABAC, but also contains related policy constraints facilitating delegations and collaborations among tenants and users in multiple levels. To facilitate attribute-based policy evaluation and implementing the proposed model, we apply an efficient mechanism to transform complex logical expressions in policies to compact decision diagrams. Our prototype of the multi-tenant access control system for Intercloud is developed, tested and integrated into the GEYSERS project [10]. Evaluations demonstrate that our system has good performance in terms of number of cloud resources, clients and policies.

2.4 Preliminaries

2.4.1 Multi-tenant Systems and Resource Ownerships

Regarding cloud resource management [79], resources in cloud are virtualized and managed in a common resource pool. Depending on the stage in its life cycle, the resource may be administrated by one or multiple entities [80], as known as the multi-tenancy pattern [5, 6, 44]:

- At the initial stage, resources are managed by the provider, who is the economic and management owner of available idle resources.
- When a tenant subscribes set of cloud resources, their economic and administrative ownerships will be transferred exclusively to this tenant during subscribed period.
- The subscribed tenant may want to allow accesses from its users, or due to collaboration requirements, share part of its resources to another trusted tenant with specific conditions like allowed actions, time, location.
- The trusted tenant in turn can manage the shared resources by defining access control policies for its users, or share to another one.

This paradigm has been discussed in different forms [33–35]. Authors in [33, 34] used the authorization tuple ⟨*issuer, subject, privilege, resource*⟩ to define the transfers from "*issuer*" to "*subject*". But the *issuer* was used only for tenants, nor the provider. They assumed the provider has its own mechanism to isolate allocated resources for different tenants, thus the model missed the isolation property between tenants at policy management level. Proposed models [33, 34] even simplified tenants collaborations with coarse-grained resolutions. This inter-tenant collaboration drawback was improved in MT-RBAC [35] when it allowed the trustee can manipulate shared roles with user assignment and role hierarchy operations. However, due to utilizing RBAC, MT-RBAC may suffer role explosion problem when more fine-grained collaborations are required, e.g., to re-share some permissions from the shared role to other tenants, or limited with environment conditions, the trustee must create new separated roles.

To solve such issues, our multi-tenant access control approach contains the following features for a cloud resource management system:

- Support diversity of subjects: different entities are able to compose policies to manage their objects, including the cloud provider, subscribed tenant and shared tenants.

- Policy generation for dynamic objects: In cloud systems, provisioned cloud resources are the objects of authorization. Their identifiers are generated during provisioning phases and released at the end of the subscription. The cloud provider needs to define authorization statements for such objects automatically according to the on-demand self-service property. Our model is defined to integrate with the resource information model, that can be used to generate policies from predefined policy templates binding with cloud service plans.

- Fine-grained access control based on the ABAC model.

- Flexible inter-tenant collaborations that allows tenants to share subscribed cloud resources in multiple levels.

- Dynamic constraints applied to policy management for multiple authorities: we define isolation and grant constraints to check if updated policies are compliant to the multi-tenant policy management properties. This guarantee would improve system performance when no conflict occurs during evaluation run-time. Previous works [33–35] are not aware of this feature.

In this chapter, firstly we introduce an information model used to manage cloud resources in our scenarios in the GEYSERS project [10] and the basic ABAC model. Based on this, we propose our model in Section 2.5.

2.4.2 Information Model for Virtual Cloud Infrastructure

Cloud resources management for both physical and virtual aspects requires a well description for modeling, discovery, composition, monitoring and synchronization. For such purposes, we use the Infrastructure and Network Description Language (INDL) [79] to model cloud resources. The ontology of IaaS cloud layer is briefly illustrated in Figure 2.1. INDL could model Virtual Resources (VRs) implemented on physical devices in different stages (e.g., abstracted, reserved and instantiated) that belong to different cloud providers. The VI is the composition built up from different types of general VRs. It enables on-demand provisioning and elastic scaling of resource capabilities.

Because the cloud information model such as INDL could represent the extensibility and flexibility of cloud resources configuration in run-time life cycles, integration of the access control with the information model permits the policies can be updated automatically upon such on-demand provisioning changes. For example, a cloud provider requires that upon subscribing an IaaS plan, the tenant could perform or manage (i.e. allow its users to act on behalf) on provisioned VMs and network links (e.g., instantiate, reconfigure, monitor), but not exceeding tenant's subscribed capabilities. This requirement can be done either by implementing directly inside the cloud management system, or decoupled the configuration with access control policies. The latter option using policies to manage capabilities of the tenant is more flexible due to any changes in the plan can reflex easily by configuration updates, not implementation changes.

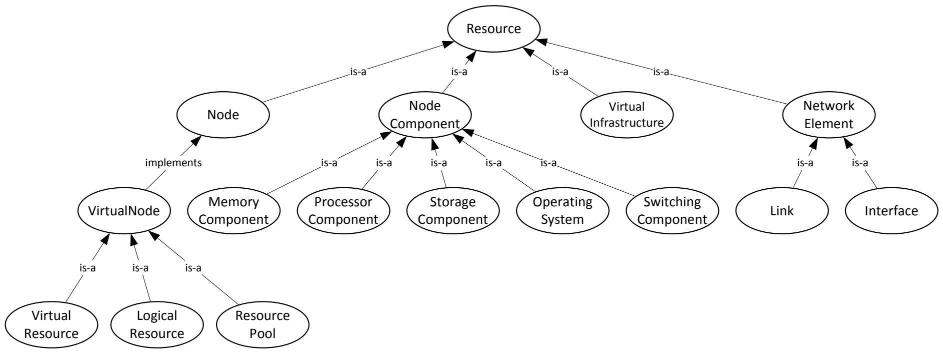


Figure 2.1: Overview of information model for cloud infrastructure resources

2.4.3 Attribute-based Access Control

ABAC definitions has been described in different forms, but in general access is determined based on matches between specific attribute values of the subject, resource and environment conditions [78, 81]. The policy implemented in ABAC is the mechanism that represents the mapping from the attribute values to authorization decisions and it is only limited by expressiveness of the computational language.

The ABAC concepts from [77, 78] will be applied in our approach:

Definition 2.1 (ABAC concepts). The ABAC has the following concepts:

- *Subject*: is the active entity to request access on a resource. In our model, the subject can either be a provider, a tenant, or a user of a tenant. A subject S is characterized by set of attributes. They may include subject's identifier, name, organization, etc. Let $A_{s_1}, A_{s_2}, \dots A_{s_n}$ be value sets (or domains) of subject's attributes, the set of subjects S is defined as the subset of the Cartesian product of k subject's attribute domains:

$$S \subseteq A_{s_1} \times A_{s_2} \times \cdots \times A_{s_k} \tag{2.1}$$

  Each subject $s \in S$ is a tuple of subject attribute values: $s = (a_{s_1}, a_{s_2}, \dots a_{s_k})$, $a_{s_i} \in A_{s_i}$, $i \in [1, k]$.

- *Resource*: is the cloud object that needs to be protected. It could either be in the idle state managed by the provider, or reservation and deployment states and managed by a tenant. A resource is referred by its identifier attribute. However, due to cloud provisioning systems, the identifier of a resource is unknown prior provisioned to the subscribed tenant. In typical ABAC models, actions on a resource often depend on the resource's characteristics. So without loss of generality, action attributes can be seen as attributes of the resources. Similar to the subject, the set of resources is defined as:

$$R \subseteq A_{r_1} \times A_{r_2} \times \cdots \times A_{r_l} \tag{2.2}$$

in which $A_{r_i}, i \in [1, l]$ is the domain of a resource's attribute. A resource $r \in R$ is a tuple of attribute values: $r = (a_{r_1}, a_{r_2}, \dots a_{r_n}), a_{r_i} \in A_{r_i}$.

- *Environment conditions*: attributes such as date, time, system security level, location, etc. are grouped as the environment attributes. The set of environment conditions is defined as:

$$E \subseteq A_{e_1} \times A_{e_2} \times \cdots \times A_{e_m} \tag{2.3}$$

  in which $A_{e_i}, i = [1, m]$ is the domain of an environment's attribute. An environment condition $e \in E$ is a tuple of attribute values: $e = (a_{e_1}, a_{e_2}, \dots a_{e_m})$, $a_{e_i} \in A_{e_i}$.

- *Authorization request*: an authorization request x is a tuple of attribute values sent by the subject entity s to the PDP asking for access to the resource r under environment condition e. The set of requests X is defined as:

$$\begin{aligned} X &= S \times R \times E \\ &= \{(s, r, e) | s \in S, r \in R, e \in E\} \end{aligned} \tag{2.4}$$

- *Policy*: attribute-based policies are represented by a policy language, that has the semantic as the first-order logic. A policy is created by an issuer that asserts the decision for given requests from subjects. It has the semantic as a predicate function mapping from authorization request domain X in Eq (2.4) to the decision domain:

$$f : X \mapsto \{Y, N\} \tag{2.5}$$

  with '*Y*' and '*N*' representing permitted and denied decisions. In this formula, the predicate function f defines a *n-ary* relation of authorization request X.

  There are different attribute-based policy languages that can be used in ABAC systems [36, 37]. In our model in section 2.5, we propose the authorization statement as the abstraction of policy. The implementation in section 2.8 discusses on how to apply XACML to transformation policies into authorization statements.

2.5 Proposed Model

2.5.1 Multi-tenant Attribute-based Access Control Model

In this section, we propose our MT-ABAC for cloud resource management. Compared to the general ABAC, it decouples subjects into providers, tenants and users of tenants, as well as defines constraints for multi-tenant management.

In our model, each cloud provider p is an autonomous system that manages a set of tenants, a set of users and a set of resources. Interactions between multiple autonomous systems (i.e. multi-providers) are discussed in Chapter 3.

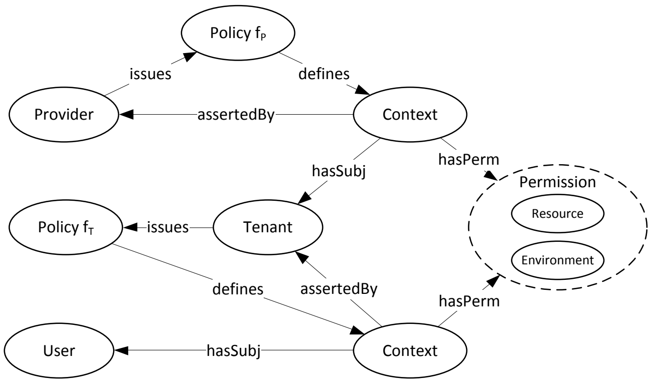


Figure 2.2: Multi-tenant access control model for cloud infrastructure resources

The Figure 2.2 represents the relationships between subjects in the cloud management systems that supports multi-tenancy properties.

The multi-tenant access control model has the following concepts:

2.5.1.1 Provider

Let $P \subseteq S$ be the set of providers who can provide resources in the multi-tenant system. A provider $p \in P$ is the subject to assign cloud resources to tenants. It is in charge of provisioning and composing the cloud resources from VRs. The cloud provider uses the information model to provision and scale cloud resources to tenants, at the same time regulate tenants' operations according to its information model.

2.5.1.2 Tenant

Denoted $T \subseteq S$ be set of tenants who can subscribe resources in the multi-tenant system. The tenant $t \in T$ is the subject to manage subscribed cloud resources with the following operations:

- Define policies to determine which of its users can access on managed resources.
- Delegate policy management to another tenant on the specified resource via delegation mechanism. This feature is used in the inter-tenant collaboration.

When a tenant subscribes resources from a provider, they have the relation $TenantOf$ which is defined as follows:

$$TenantOf \subseteq T \times P \tag{2.6}$$

With this relation, the set of tenants of the provider $p \in P$ is denoted as $T(p)$ or T_p:

$$T(p) = \{t \in T | t\ TenantOf\ p\}$$

2.5.1.3 User

Let $U \subseteq S$ be the set of all users who consumes resources in the multi-tenant system. In our model, P, T and U have the following properties:

$$S = P \cup T \cup U$$
$$(P \cup T) \cap U = \emptyset$$

The relation $UserOf$ defines the set of users of a tenant:

$$UserOf \subseteq U \times T \tag{2.7}$$

Given a tenant $t \in T$, its users is denoted as $U(t) = \{u \in U | u\ UserOf\ t\}$.

The set of users of a provider p is denoted as:

$$U(p) = \bigcup_{t \in T(p)} U(t) \tag{2.8}$$

2.5.1.4 Resource

Let R be set of all resources as in equation (2.2). A set of resources owned by either a tenant or provider defined by the relation:

$$ResourceOf \subseteq R \times (T \cup P) \tag{2.9}$$

A tenant $t \in T$ has its subscribed resource $R(t) = \{r \in R | r\ ResourceOf\ t\}$. A provider $p \in P$ manages their idle resources $R(p) = \{r \in R | r\ ResourceOf\ p\}$. According to the exclusive resource ownership in multi-tenant systems, $\forall x, y \in T \cup P$ we have:

$$R(x) \cap R(y) = \emptyset \Leftrightarrow x \neq y \tag{2.10}$$

2.5.1.5 Permission

Let $\mathcal{P}$ be the set of permissions defined as:

$$\mathcal{P} \subseteq R \times E \tag{2.11}$$

Each permission is a tuple $(r, e) \in \mathcal{P}$ represented by a set of attributes identifying a resource with its action, and specific environment condition attributes.

2.5.1.6 Context

Definition 2.2 (authorization statement). is the assertion to say that the issuer $i \in S$ authorizes subject $s \in S$ on a permission (r, e). It is denoted as $authz(i, s, (r, e))$.

The authorization statement has the transitive property [69] as follows:

$$authz(s_1, s_2, (r, e)) \wedge authz(s_2, s_3, (r, e)) \rightarrow authz(s_1, s_3, (r, e)) \tag{2.12}$$

Definition 2.3 (Context). is a statement of the issuer $i \in S$ on the approval of set of permissions to the subject $s \in S$. A statement can be denoted by a tuple $(issuer, subject, \{permissions\})$, in which the *issuer* can be either a provider or a tenant and the *subject* can be either a tenant or a user. The formal definition of the context is described in equation (2.15).

We classify the following contexts:

- Authorization context (AC): is issued by a tenant to a user:

$$AC \subseteq T \times U \times \mathcal{P}^n \tag{2.13}$$

- Delegation context (DC): is issued by either a provider to its tenant or a tenant to another tenant:

$$DC \subseteq (P \cup T) \times T \times \mathcal{P}^n \tag{2.14}$$

Formally, the context is defined as:

$$\begin{aligned} C &= AC \cup DC \\ &\subseteq (T \times U \cup (P \cup T) \times T) \times \mathcal{P}^n \end{aligned} \tag{2.15}$$

Given a context $c \in C$, we denote $\mathcal{I}(c)$ as the issuer of c, $\mathcal{S}(c)$ as the subject of c and $\mathcal{P}$(c) as the set of permissions in c.

According to definition 2.2, given a context $c = (i, s, \mathcal{P}(c))$, we have:

$$\forall (r, e) \in \mathcal{P}(c), authz(i, s, (r, e)) \tag{2.16}$$

Let $\mathcal{R}(c)$ be set of resources referred in the context c. Formally, it is defined as:

$$\mathcal{R}(c) = \{r | r \in R, \exists (r, e) \in \mathcal{P}(c)\} \tag{2.17}$$

2.5.1.7 Authorization Request

The request $x = (s, r, e) \in X$ is defined in equation (2.4).

Given a context c, the request x is authorized by a c if and only if:

$$\begin{cases} s = \mathcal{S}(c) \\ (r, e) \in \mathcal{P}(c) \end{cases} \tag{2.18}$$

According to (2.16), we see that it is equivalent to the authorization statement $authz(\mathcal{I}(c), s, (r, e))$.

The multi-tenant systems have multiple authorities in which each can define policies freely. According to equation (2.15) in our model, providers and tenants can create contexts (i.e. define policies). So it is possible that some contexts from different issuers may be conflicted and their decisions are contradicted which makes the system inconsistent. In our MT-ABAC, we resolve this problem by defining the delegation model, trusted contexts and constraints. They guarantee that at a given state, the system always returns the consistent decision for an authorization request.

2.5.2 Delegations in MT-ABAC

2.5.2.1 Types of Delegation Contexts

From equations (2.15), our delegation model defines relations between authorities in the MT-ABAC , including providers, tenants and users of tenants as follows:

- Providers can issue delegation contexts to tenants.
- A tenant can issue delegation contexts to other tenants.
- A tenant can issue authorization contexts to users.

According to delegation categories classified in [68, 69], we distinguish delegation contexts based on types of issuers and subjects as follows:

- *Transfer context* (TC): when a provider provisions its cloud resources to a tenant, it uses the transfer context in which resources are exclusively allocated to only this tenant. The tenant and the provider then have the relationship *TenantOf* as in equation (2.6). The set of transfer contexts is defined as:

$$TC = \{(x, y, \{(r, e)\}) | x \in P, y \in T(x), (r, e) \in \mathcal{P}\} \tag{2.19}$$

- *Grant context* (GC): when a tenant want to share a part of its resources to another tenant, it creates a grant context.

$$GC = \{(x, y, \{(r, e)\}) | x, y \in T \setminus P, (r, e) \in \mathcal{P}\} \tag{2.20}$$

  It shows that only tenants without having provider role can create grant contexts.

From the equations (2.19) and (2.20), we have the following properties:

$$TC \cap GC = \emptyset \tag{2.21}$$
$$DC = TC \cup GC \tag{2.22}$$

When transferring a resource $r \in R$ from the provider to a tenant $t \in T$, it creates the relation $ResourceOf$ as in (2.9) between r and t. So the total resource owned by a tenant $t \in T$ is:

$$R(t) = \bigcup_{\forall c \in TC, \mathcal{S}(c) = t} \mathcal{R}(c) \tag{2.23}$$

Based on the accountable properties in cloud, in which a subscribed cloud resource is exclusively assigned to a tenant during a definite lifetime, we need to define a constraint on transfer contexts so that a resource cannot be provisioned to more than one tenant at a specific environment condition. The isolation constraint will be defined in section 2.5.3.

The multi-tenancy system allows tenants to collaborate via the inter-tenant operations, i.e. a resource of a tenant can be accessed by either users of this tenant, or users of another tenant. The inter-tenant is supported by grant contexts as described above. However, to guarantee that a tenant cannot create grant contexts for resources it does not have permissions, we define the grant constraint in section 2.5.3.

2.5.2.2 Context Relationships

To solve conflicting issues may arise for multiple authorities, we present relationships between issued contexts. At first, they are described for a single provider, then are extended to multiple providers.

Definition 2.4 (Provider's trust contexts). In a MT-ABAC system of a provider $p \in P$ with a set of its tenants $T(p)$ and their users $U(p) = \bigcup_{t \in T(p)} U(t)$, let $\mathcal{C}(p)$ be set of contexts trusted by p. A context $c = (i, s, \mathcal{P}(c))$ is trusted by p (a.k.a $c \in \mathcal{C}(p)$) if and only if:

$$\forall (r, e) \in \mathcal{P}(c), authz(p, i, (r, e)) \tag{2.24}$$

Lemma 1. If c^* is a transfer context made by p, it is trusted by the provider p:

$$(c^* \in TC) \wedge (\mathcal{I}(c^*) = p) \rightarrow c^* \in \mathcal{C}(p) \tag{2.25}$$

Proof: Because c^* is the transfer context, we have $\mathcal{I}(c^*) = p$. According to (2.16):

$$\forall (r, e) \in \mathcal{P}(c^*), authz(p, \mathcal{S}(c^*), (r, e))$$

So by trust context definition (2.24), we conclude c^* is trusted by p, or $c^* \in \mathcal{C}(p)$. The set of all trusted contexts in the MT-ABAC is denoted as:

$$\mathcal{C} = \bigcup_{\forall p \in P} \mathcal{C}(p) \tag{2.26}$$

To manage trust contexts of a provider efficiently, we define the partial relationship between two contexts:

Definition 2.5 (Partial trust relationship). The partial trust relationship $PT \subseteq C \times C$ over two trust contexts of provider p is defined as:

$$PT = \{(c_i, c_j) | c_i, c_j \in \mathcal{C}(p), \mathcal{S}(c_i) = \mathcal{I}(c_j) \wedge \mathcal{P}(c_i) \cap \mathcal{P}(c_j) \neq \emptyset\} \tag{2.27}$$

Thus the function $PT(c)$ returns all contexts in $\mathcal{C}(p)$ that c partially trusts. Formally:

$$PT(c) = \{c' \in \mathcal{C}(p) | (c, c') \in PT\} \tag{2.28}$$

Definition 2.6 (Tenant privilege scope). the tenant $t \in T$ has its privileges scope:

$$\overline{\mathcal{P}}(t) = \bigcup_{\forall c \in \mathcal{C}, \mathcal{S}(c) = t} \mathcal{P}(c) \tag{2.29}$$

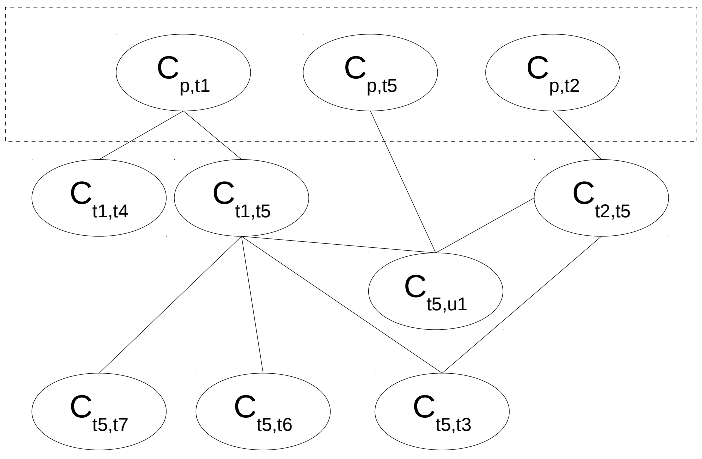


Figure 2.3: An example of context relationships

The MT-ABAC delegations allow a tenant t to share its resources to another one via grant contexts. However, these contexts are not always trusted, because t may grant permissions on unowned resources. To identify if these contexts are trusted, the system need to check if all their permissions belong to the privilege scope of t. If a tenant can grant any contexts, the system runtime overhead on checking their trusts is costly. In section 2.5.3, we define constraints at the policy composition stage to prevent such overheads on runtime stage. These constraints can improve the system performance.

The Figure 2.3 presents context trees of the provider p. Transfer contexts c_{p,t_1}, c_{p,t_2} and c_{p,t_5} are trusted directly by the provider p. Each connection represents the trust relationship between contexts. A context can be trusted by only one context or multiple contexts. In the figure, tenant t_1 share some resources directly from its transfer context to t_4, so $\mathcal{P}(c_{t_1,t_4}) \subseteq \mathcal{P}(c_{p,t_1})$. In the other case, the context c_{t_5,u_1} created by t_5 that combined permissions from different contexts c_{p,t_5}, c_{t_1,t_5} and c_{t_2,t_5} so c_{t_5,u_1} is partially trusted by those contexts.

2.5.3 Multi-tenancy Constraints

The system state of a provider $p \in P$ contains the following information: $\mathcal{C}(p)$, $T(p)$ and $U(p)$. To simplify definitions, we shorten the system state as $(\mathcal{C}_p, T_p, U_p)$. It evolves over time via administrative operations from the provider and its tenants. In this section, we define constraints to make sure the MT-ABAC system state is consistent.

2.5.3.1 Isolation Constraint

The multi-tenancy systems require that tenants should have security isolation [75] on different layers. It can be achieved by using implicit filter based on the binding between tenant-id and allocated resources, or with explicit permission-based access control isolation. Our work support the later at the conceptual level by the isolation constraint below.

To guarantee the exclusive resource ownership property in the multi-tenant system, the equation (2.10) should be satisfied. Given $x, y \in T$, from (2.23) and (2.10), we have:

$$\forall c_1, c_2 \in TC, (\mathcal{S}(c) = x) \wedge (\mathcal{S}(c_2) = y) \rightarrow \mathcal{R}(c_1) \cap \mathcal{R}(c_2) = \emptyset \tag{2.30}$$

The equation (2.30) guarantees that the provider does not transfer a resource to more than one tenant. From this condition, we define the isolation constraint as follows:

Definition 2.7 (Isolation constraint). Given a system state $(\mathcal{C}_p, T_p, U_p)$, the constraint $canTransfer(c^*)$ on a given transfer context $c^* \in TC$ is valid iff:

$$\forall tc' \in \{tc \in \mathcal{C}_p | \mathcal{I}(tc) = p\}(\mathcal{I}(c^*) = p \wedge \mathcal{S}(c^*) \in T_p \wedge \mathcal{R}(c^*) \cap \mathcal{R}(tc') = \emptyset) \tag{2.31}$$

in which $\mathcal{R}(c)$ are sets of resources referred in contexts c that is defined in equation (2.17).

2.5.3.2 Grant Constraints

In the Figure 2.3, it is possible that the tenant t_5 issues to t_3 a context containing out-of-scope privileges of t_5 itself (e.g., the write permission on a t_1's folder, while t_1 only allows t_5 to read from it). Therefore, the context c_{t_5,t_3} is untrusted. If there are many untrusted contexts in practical, the authorization process becomes more complex, that affects the system performance.

To prevent this problem, we define the grant constraint applying to the policy composition of tenants (i.e. when a tenant adds, removes or updates its policies) to make sure no context is untrusted. Although this constraint will increase policy composition checking overhead, the authorization evaluation process of the system will improve.

Definition 2.8 (Grant constraints). Given a system state $(\mathcal{C}_p, T_p, U_p)$ and a context c^* with $i^* = \mathcal{I}(c^*), s^* = \mathcal{S}(c^*)$, the grant constraints are defined as:

- If $c^* \in AC$: the constraint $canGrantAC(c^*)$ is valid iff:

$$(i^* \in T_p) \wedge (s^* \in U(i^*)) \wedge (\mathcal{P}(c^*) \subseteq \overline{\mathcal{P}}(i^*)) \tag{2.32}$$

- If $c^* \in GC$: the constraint $canGrantGC(c^*)$ is valid iff:

$$(i^* \in T_p) \wedge (s^* \in T_p) \wedge (i^* \neq s^*) \wedge (\mathcal{P}(c^*) \subseteq \overline{\mathcal{P}}(i^*)) \tag{2.33}$$

For a given context $c^* \in AC \cup GC$, the grant constraint can be written as $canGrant(c^*)$.

The complexity of the algorithm to determine if the context c meets the grant constraint depends on the subset checking operation $\mathcal{P}(c^*) \subseteq \overline{\mathcal{P}}(i^*)$.

Lemma 2. Given a system state $(\mathcal{C}_p, T_p, U_p)$ and a context $c^* \in AC \cup GC$. The context c^* is trusted by the provider p if and only if it satisfies the grant constraints.

$$(c^* \in AC \cup GC) \wedge canGrant(c^*) \leftrightarrow c^* \in \mathcal{C}(p) \tag{2.34}$$

Proof: For the "if" direction: the context c^* can either be an authorization context or grant context. In both cases, we have $\mathcal{P}(c^*) \subseteq \overline{\mathcal{P}}(i^*)$. So that:

$$\forall (r,e) \in \mathcal{P}(c^*) \rightarrow (r,e) \in \overline{\mathcal{P}}(i^*)$$

According to (2.29):

$$\forall (r,e) \in \mathcal{P}(c^*), \exists c' \in \mathcal{C}(p), ((r,e) \in \mathcal{P}(c')) \wedge (\mathcal{S}(c') = i^*) \tag{2.35}$$

From the definition of the trust context (2.24) applied to c', we have:

$$c' \in \mathcal{C}(p) \rightarrow \forall (r,e) \in \mathcal{P}(c'), authz(p, \mathcal{I}(c'), (r,e)) \tag{2.36}$$

However, based on the definition of the authorization context (2.16), we have:

$$\forall (r,e) \in \mathcal{P}(c'), authz(\mathcal{I}(c'), \mathcal{S}(c'), (r,e)) \tag{2.37}$$

Using the transitive property (2.12), from (2.36) and (2.37), we have

$$\forall (r,e) \in \mathcal{P}(c'), authz(p, \mathcal{S}(c'), (r,e)) \tag{2.38}$$

Replace $i^* = \mathcal{S}(c')$ to (2.38) and combine with (2.35):

$$\forall (r,e) \in \mathcal{P}(c^*), authz(p, i^*, (r,e)) \tag{2.39}$$

Based on the trust context definition (2.24) applied to (2.39):

$$\forall (r,e) \in \mathcal{P}(c^*), authz(p, i^*, (r,e)) \rightarrow c^* \in \mathcal{C}(p) \tag{2.40}$$

For the "only if" direction: according to privilege scope definition (2.29):

$$\overline{\mathcal{P}}(i^*) = \bigcup_{\forall c \in \mathcal{C}(p), \mathcal{S}(c) = i^*} \mathcal{P}(c)$$

Because $c^* \in \mathcal{C}(p)$, we have $\mathcal{P}(c^*) \subseteq \overline{\mathcal{P}}(i^*)$.

If c^* is an authorization context, we have $(i^* \in T_p) \wedge (s^* \in U(i^*))$, so the $canGrantAC(c^*)$ is valid.

If c^* is a grant context, we have $(i^* \in T_p) \wedge (s^* \in T_p)$, so the $canGrantGC(c^*)$ is valid.

In other words, the $canGrant(c^*)$ is valid.

2.5.4 MT-ABAC Operations

We differentiate two phases in the access control: policy composition phase and authorization evaluation phase. The first occurs when the provider allocates resources to tenants or a tenant composes policies for its users. The later is for authorization request evaluations from users to consume cloud resources.

The system state $(\mathcal{C}_p, T_p, U_p)$ of a provider $p \in P$ evolves over times via administrative commands from the provider p and its tenants T_p. Assume that $(\mathcal{C}'_p, T'_p, U'_p)$ is the new state after an administrative command, the Table 2.1 summaries these commands as follows:

Table 2.1: Administrative commands for MT-ABAC system

Command	Condition	Update
$addTenant(t)$	$t \notin T_p$	$T'_p = T_p \cup \{t\}$
$removeTenant(t)$	$\nexists c \in \mathcal{C}_p(\mathcal{I}(c) = t \vee \mathcal{S}(c) = t)$	$T'_p = T_p \setminus \{t\}$
$addUser(t, u)$	$t \in T_p \wedge u \notin U(t)$	$U'(t) = U(t) \cup \{u\}$
$removeUser(t, u)$	$t \in T_p \wedge \nexists c \in \mathcal{C}_p(\mathcal{S}(c) = u)$	$U'(t) = U(t) \setminus \{u\}$
$transfer(tc)$	$tc \in TC \wedge canTransfer(tc)$	$\mathcal{C}'_p = \mathcal{C}_p \cup \{tc\}$
$grant(c)$	$(c \in AC \cup GC) \wedge canGrant(c)$	$\mathcal{C}'_p = \mathcal{C}_p \cup \{c\}$
$removeContext(c)$	$c \in \mathcal{C}_p$	$removeContext(\mathcal{C}_p, c)$

The algorithm 2.1 is to remove a context c and update $\mathcal{C}_p$.

```
proc removeContext($\mathcal{C}_p, c$)
    updateTrustCtxs($\mathcal{C}_p, c$)
    $\mathcal{C}_p \leftarrow \mathcal{C}_p \setminus \{c\}$
    return
proc updateTrustCtxs($\mathcal{C}_p, c$)
    foreach ($c' \in PT(c)$) do
        $perms \leftarrow \mathcal{P}(c') \setminus \mathcal{P}(c)$
        if ($perms \neq \emptyset$) then
            $c' \leftarrow (\mathcal{I}(c'), \mathcal{S}(c'), perms)$
            updateTrustCtxs($\mathcal{C}_p, c'$)
        else
            $\mathcal{C}_p \leftarrow \mathcal{C}_p \setminus \{c'\}$
        end
    end
    return
```

Algorithm 2.1: Remove a context and update $\mathcal{C}_p$ in MT-ABAC

In the authorization evaluation phase, the MT-ABAC evaluates a request x from a user by finding an authorization context in $\mathcal{C}$ so that the request x is authorized by c as defined in equation (2.18).

2.6 Analysis

In this section, we prove that our system is consistent: given a system state $(\mathcal{C}_p, T_p, U_p)$, after any administrative operations, the new system state $(\mathcal{C}'_p, T'_p, U'_p)$

always maintains the property that all contexts in $\mathcal{C}'_p$ are trusted:

$$\forall c \in \mathcal{C}'(p), \forall(r,e) \in \mathcal{P}(c), authz(p, \mathcal{I}(c'), (r,e)) \tag{2.41}$$

For the operations $addTenant, removeTenant, addUser, removeUser$, the set of contexts does not change: $\mathcal{C}'_p = \mathcal{C}_p$, so (2.41) is valid.

For the $transfer(tc)$ operation, $\mathcal{C}'_p = \mathcal{C}_p \cup \{tc\}$. Because $tc \in TC$, according to Lemma 1, tc is trusted by p, and (2.41) is valid.

For the $grant(c)$ operation, $\mathcal{C}'_p = \mathcal{C}_p \cup \{c\}$. According to Lemma 2, c is trusted by p then (2.41) is also valid.

For the $removeContext(c^*)$ operation, the algorithm 2.1 is to recalculate all contexts c that partially trusted by c^*. It removes the common permissions shared between c and c^*, so makes c^* and c do not have any relationship: $c \notin PT(c^*)$. The updated context of c therefore is still trusted by p. The updating process continues recursively until no context in $\mathcal{C}(p)$ has any share permission with c^* or $PT(c^*) = \emptyset$. So this algorithm guarantees that all updates contexts are trusted by p.

The consistency of our system guarantees that in the authorization phase, given a system state $(\mathcal{C}_p, T_p, U_p)$ and an authorization request $x = (s, r, e)$, the evaluation process is valid:

$$\exists c^* \in \mathcal{C}_p, authz(\mathcal{I}(c^*), s, (r,e)) \vdash authz(p, s, (r,e))$$

The proof is simple. Because $c^* \in \mathcal{C}_p$, from definition (2.24), we say that $authz(p, \mathcal{I}(c^*), (r,e))$. Using the transitive property in (2.12), we conclude:

$$authz(\mathcal{I}(c^*), s, (r,e)) \wedge authz(p, \mathcal{I}(c^*), (r,e)) \rightarrow authz(p, s, (r,e))$$

2.7 Integration MT-ABAC with INDL

2.7.1 Attribute-based Policy Semantic Model

In the MT-ABAC system for clouds, the provider's policies need to be generated automatically based on on-demand resource provisioning, while the tenants' policies can be configured later by customers. Thus, we integrate the MT-ABAC into the cloud information model INDL [79] shown in the Figure 2.4, which represents provider's policies. Tenants can use other ABAC languages like XACML to compose their policies without limitation.

Any authorization requests to the resource must be checked against these attached policies.

In this INDL extension, the provider's policy is bounded to the resource concept via the relationship *hasPolicy*. Because of the inheritance of resource types, derived resources have all policies of their ancestors, leading conflicting may arise. Thus, we define combining operators for joining such multiple attached policies. However, rather than many combining operators as in XACML where multiple parties can create conflicting policies, we only need *permit-override* for privilege enrichment

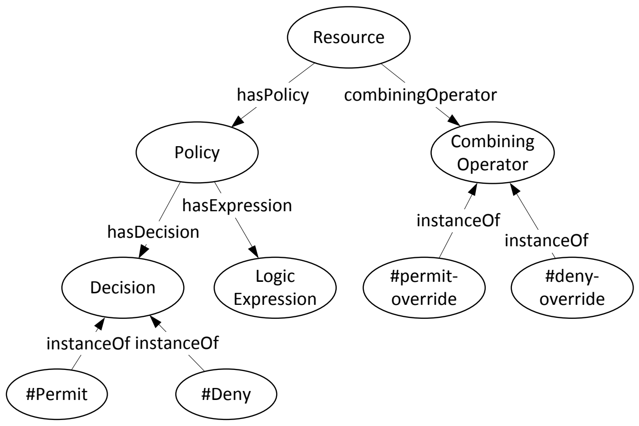


Figure 2.4: Attribute-based policy model integration with INDL

and *deny-override* for privilege limitation for inheritance relationships: e.g., the ancestor VirtualNode concept allows to add a new network interface, and its descendant node, a VM, allow to instantiate, so the *permit-override* operator can be used in this case. Thus, it can be seen that the provider's policies in our MT-ABAC utilize a subset of XACML.

In cloud resource life-cycles, the reservation and instantiation of resources occur automatically based on tenants' requests. We need a mechanism to create authorization rules to manage these on-demand resources:

- We define policies for each resource components in a VI, which forms a policy template.
- The policy template is then used to generate the instantiated policies for deployed cloud resources.

2.7.2 Policy Generation from Cloud Infrastructure Descriptions

In the Figure 2.2, the provider issues policies for tenants to delegate permissions on their subscribed resources. These operations should be performed automatically in the cloud resource life-cycles, i.e. issuing, updating and revoking policies at the reservation, re-planning and decommissioning phases, respectively. We propose an automatic mechanism to generate policies from cloud infrastructure resource descriptions using INDL as follows:

- Given an infrastructure resource description using INDL implemented by RDF/OWL technologies, we use SPARQL queries to obtain detail resource information of the cloud resources, including resource identifiers, types, owner, as in Listing 2.1.
- We define the resource policy template using model in Figure 2.4, which specifies policies could bind to a resource type, as illustrated in Figure 2.5. The policies are retrieved as in Listing 2.2.

- We create policies with the subject be the tenant, resource identifiers and types along with related actions from the template. They can be either expressed as the generic attribute-based policy notation or a policy language standard like XACML [36].

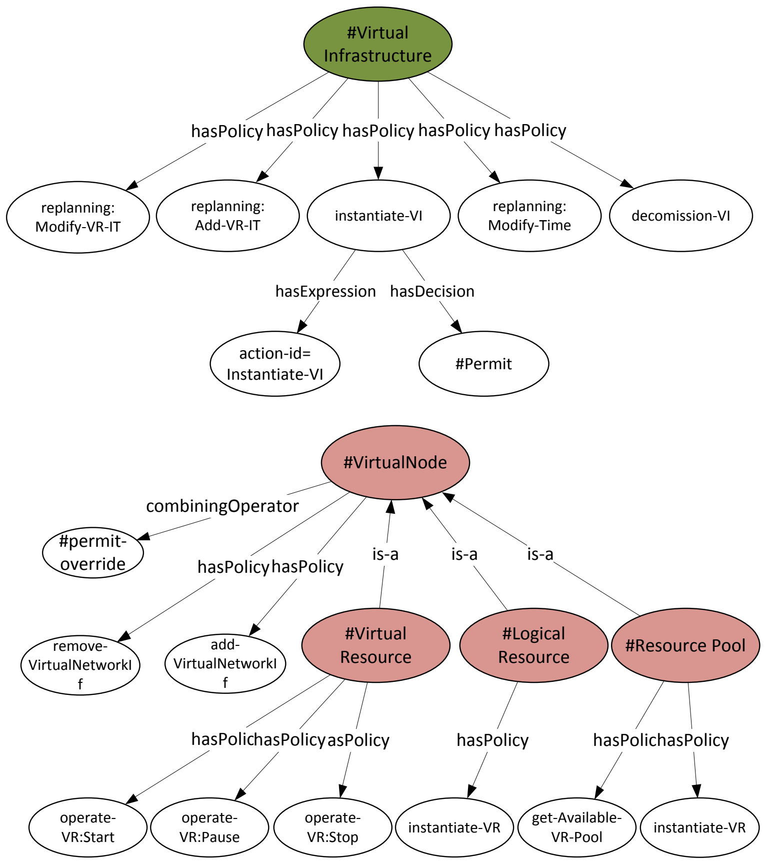


Figure 2.5: Defining policy template sample

```
PREFIX rdfs: <http://www.w3.org/2000/01/
rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-
rdf-syntax-ns#>
PREFIX imf: <http://geysers.eu/imf.owl#>
SELECT ?vi ?r ?rtype
WHERE {
        ?vi rdf:type imf:VirtualInfrastructure.
        ?vi imf:hasResource ?r.
        ?r rdf:type ?rtype.
        ?rtype rdfs:subClassOf* imf:Resource.
}
```

Listing 2.1: Query resource components in the virtual infrastructure

```
PREFIX rdf: <http://www.w3.org/1999/02/
 22-rdf-syntax-ns#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/
 rdf-schema#>
PREFIX imf: <http://geysers.eu/imf.owl#>
PREFIX daci: <http://geysers.eu/imf-daci.owl#>
SELECT ?r ?type ?cop ?d ?expr
WHERE {
        #resourcetype# rdfs:subClassOf* ?type.
        ?r rdf:type ?type.
        ?r daci:combiningOperator ?cop
        ?r daci:hasPolicy ?p.
        ?p daci:hasDecision ?d.
        ?p daci:hasExpression ?expr
}
```

Listing 2.2: Retrieve actions for a resource type from its ancestors

In Listing 2.2, the *resourcetype* parameter is a *Resource* concept in the INDL ontology. Queried result is translated into XACML policies with the equivalent combining operator and logic expressions. These providers' policies are then transformed into the data structure representing root contexts in the next section.

2.8 Mechanism to Manage Contexts in MT-ABAC

In cloud systems with high-scale of resources and tenants, approaches using attribute-based policies such as XACML [36, 38] have the advantage of high expressiveness of policy composition. However, there's no efficient mechanism in term of performance to evaluate and manage these policies. Authors in [35] used traditional XACML implementation [82] with limited results. Approaches in [33, 34] use SWRL and a DL reasoner engine to evaluate policies, which are not mainly designed as an authorization policy engine.

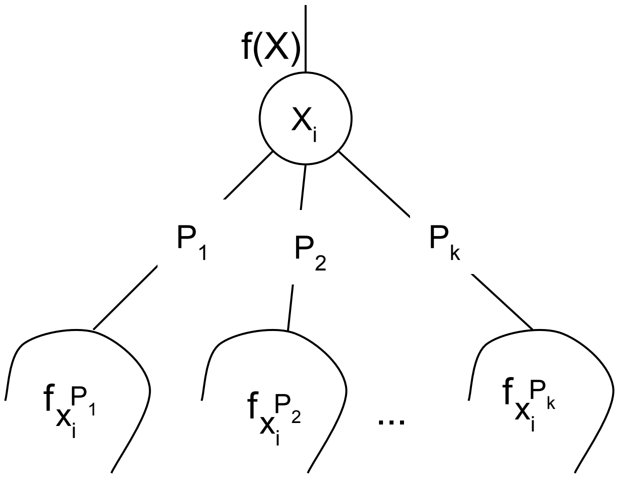


Figure 2.6: A sample Boole-Shannon decision diagram

In [83], we have proposed a new mechanism called Multi-datatype Interval Decision Diagram (MIDD) to solve such issues on the expressive attribute-based policies. Detail formulations and definitions of MIDD and Multi-datatype Interval Decision Diagram for XACML (X-MIDD) data structures can be found in chapter 4. In this section, we apply the MIDD mechanism to manage contexts of the MT-ABAC model.

2.8.1 Decision Diagrams

According to the Boole-Shannon expansion, a multi-variable logical function $f : D_1 \times D_2 \ldots \times D_n \to Boolean$ can be decomposed to partial functions which are free from a variable x_i:

$$f(X) = \bigvee_{P \in \mathcal{P}(D_i)} h_{x_i}(P) \wedge f_{x_i^P} \tag{2.42}$$

in which $h_{x_i}(P)$ represents a function returning 1 if $x_i \in P$, otherwise 0. P is a partition range of variable x_i. This function can be represented by a decision diagrams as in Figure 2.6:

However, the XACML language that we apply in MT-ABAC has more complex function signatures. The Match, AllOf, AnyOf and Target elements have the signature in Eq. (2.43), while Rule, Policy and Policyset elements have the signature in Eq. (2.44).

$$f : D_1 \times D_2 \ldots \times D_n \to V_M \tag{2.43}$$

with V_M is the domain of match values from Table 4.3:

$V_M = \{T, F, IN\}$

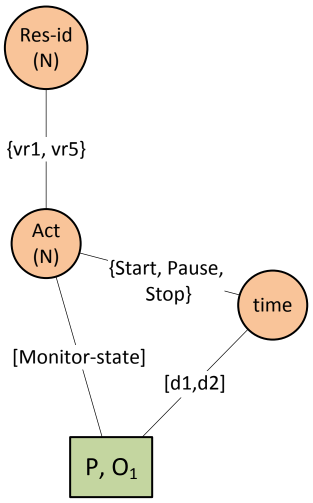


Figure 2.7: X-MIDD representing authorization statements

$$f : D_1 \times D_2 \ldots \times D_n \rightarrow V_R \tag{2.44}$$

The V_R is the decision rule domain as in Table 4.4:
$V_R = \{P, D, N, IN_P, IN_D, IN_{PD}\}$

We define extended decision diagrams as MIDD and X-MIDD representing Eq. (4.24) and Eq. (4.25) respectively.

The MIDD can express a XACML match component, while the X-MIDD represents a rule or policy. In [47], we developed algorithms to transform XACML policies into these data structures for evaluation purpose. The X-MIDD in this chapter can be used as the basis for the context object in MT-ABAC model. A X-MIDD example is illustrated in Figure 2.7.

2.8.2 Context Structure

In Section 2.5, we propose the context concept representing authorization statements of policies. In this section, we use X-MIDD as the mechanism to implement this concept.

According to definitions in Section 2.5.1, a context c contains the issuer $\mathcal{I}(c)$, the subject $\mathcal{S}(c)$ and set of permissions $\mathcal{P}(c)$. The issuer can either be the provider or a tenant, the subject identifies a tenant or a user. Set of permissions $\mathcal{P}(c)$ contains a set of tuples (r, e) to indicate which resource can be touched (the r) in the equivalent condition (e). We define a context data structure that have:

- Issuer identifier: is an attribute value or set of attribute values referring to the provider or tenants. In our attribute profile for INDL, the provider and tenants have their unique identifiers, so they can be stored here.

- Subject identifier: contains set of subject attributes (e.g., subject-id, subject-role in XACML attribute profile).
- Permissions: are stored in a X-MIDD structure. It has its variable order, in which subject attributes are at the high level in the tree, resource and environment attributes are at deeper levels, leaf nodes contains *permit* decisions. The X-MIDD has multiple paths from root to its leaf nodes, each path is an authorization statement.

We use XACML as the policy language for tenants, and a subset of XACML as the policy language of the provider. It is also possible to transform and combine a XACML policy-tree of the provider or a tenant into a X-MIDD [47]. Then the result X-MIDDs can represent issued contexts. We need to manage these contexts using constraints in Section 2.5.3.

2.8.3 Operations

According to Section 2.5, the context has the following operations:

- Function $canTransfer(tc)$: it implements the equation (2.31)
- Functions $canGrantAC(c)$, $canGrantGC(c)$ in formulas (2.32) and (2.33) are illustrated in the algorithm 2.2.
- Request authorization: given a request x and a context c, check if c authorizes x. It can be done by traveling from the root of the context's X-MIDD, if it can reach the leaf node, then the request is authorized.

2.8.4 Complexities

Given a system with $|T|$ tenants, each defines two policy-trees, one for its users called intra-tenant policy-tree and the other for tenant sharing called inter-tenant policy-tree. The provider p issues policies to tenants, which can be combined to a single policy-tree. Because each policy-tree can be transformed into a X-MIDD [47], the size of the trusted contexts $\mathcal{C}$ is $(2|T|+1)$. The complexity of the algorithm $canGrant$ in the worst case is $\mathcal{O}((2|T|+1).|c|)$ with $|c|$ is the number of paths from the root of X-MIDD in the context c.

2.9 System Design

2.9.1 DACI Architecture and Integration

We design the Dynamic Access Control Infrastructure (DACI) as in Figure 2.8 in collaboration with an Intercloud architecture in GEYSERS project [10]. In this architecture, the VIP could utilize and aggregate computing, storage and network resources from set of PIPs to compose VI services for tenants, known as VIOs. The INDL [79] is used to model and share VRs description between entities.

```
function canGrant($c$)
    foreach $((r, e) \in \mathcal{P}(c))$ do
        $found \leftarrow false$;
        $it \leftarrow \mathcal{C}_p.iterator$;
        while $((\neg found \wedge it.hasNext()))$ do
            $c' \leftarrow it.getNext()$;
            if $(\mathcal{S}(c') = \mathcal{I}(c) \wedge (r, e) \in \mathcal{P}(c')$ then
                $found \leftarrow true$;
            end
        end
        if $(\neg found)$ then
            return $false$;
        end
    end
    return $true$
function canGrantAC($c$)
    $i \leftarrow \mathcal{I}(c)$
    return $i \in T_p \wedge \mathcal{S}(c) \in U(t) \wedge$ canGrant($c$)
function canGrantGC($c$)
    $i \leftarrow \mathcal{I}(c)$
    $s \leftarrow \mathcal{S}(c)$
    return $i \in T_p \wedge s \in T_p \wedge i \neq s \wedge$ canGrant($c$)
```

Algorithm 2.2: Grant constraint functions

Each PIP has an instance of Open Nebula [84] to manage its VRs. PIP runs a system (known as Lower-Logical Infrastructure Composition Layer (LICL)) operating on top the OpenNebula instance via adapters [10] to abstract, control and monitor virtual resource information. This information is synchronized to the Upper-LICL system at VIP. The VI composed by VIP could be distributed across different PIP domains, while the tenant (VIO) of a VI can manage it via the VIP as follows:

- The VIO can send a VI request to the VIP, where the request is analyzed. Based on current available free resources from registered PIPs, the request is broken down into parts which will be provisioned at PIPs. The DACI handles the VI request in reservation phase by the *TenantManagement Service*, that generates provider's delegation policies for a given request. These policies must satisfy the isolation constraint according to Section 2.5.

- Once the VI is deployed, the VIO can define authorization policies for its end-users via the *TenantAdmin Service*. It also can set which parts of the deployed VI are shared with other tenants via the tenant's delegation policies. These policies are composed in XACML.

- The VIO and its end-users can control VI components via VR Proxies of Upper-LICL, where authorization interceptors extract request attributes and authorize at DACI against local tenant's authorization policies, inter-tenant policies and provider's delegation policies. Depending on returned decisions, the interceptors may reject or permit to process requests.

While the DACI services are designed to integrate with cloud management systems in the GEYSERS project [10], it is also possible to use DACI in other cloud

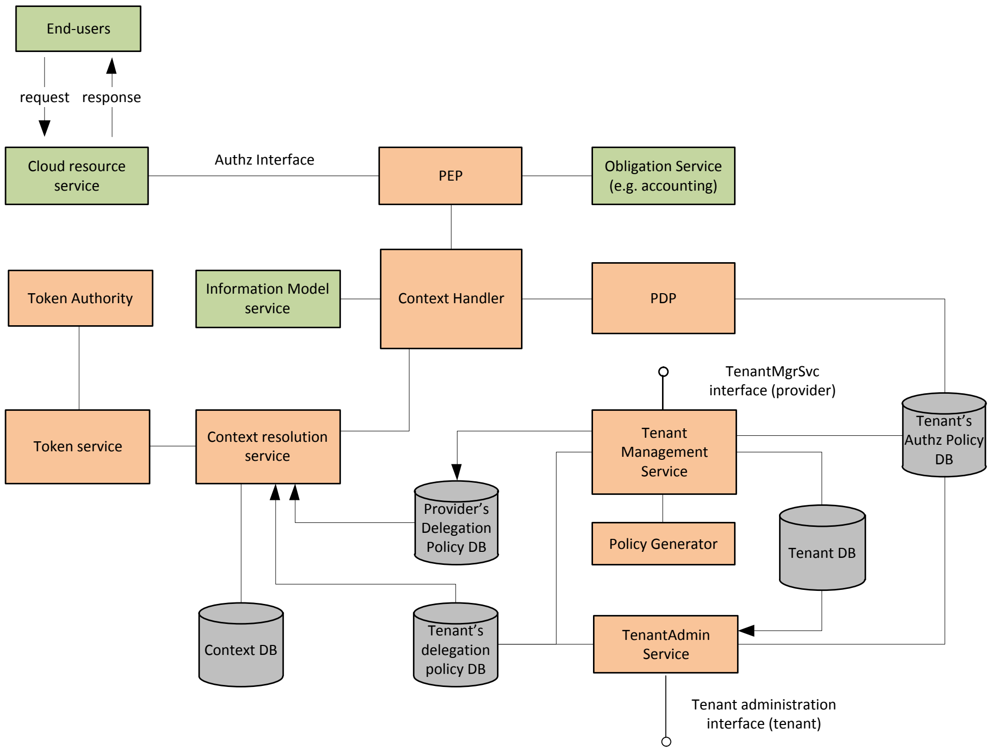

End-users
request
response
Cloud resource service
Authz Interface
PEP
Obligation Service (e.g. accounting)
Token Authority
Information Model service
Context Handler
PDP
TenantMgrSvc interface (provider)
Token service
Context resolution service
Tenant Management Service
Tenant's Authz Policy DB
Provider's Delegation Policy DB
Policy Generator
Tenant DB
Context DB
Tenant's delegation policy DB
TenantAdmin Service
Tenant administration interface (tenant)

Figure 2.8: Dynamic Access Control Infrastructure using MT-ABAC model

management systems by means of equivalent cloud resource description models. In such cases, the policy generator component allows to parse cloud resource descriptions. The integration APIs with a cloud management system are illustrated in Table 2.2.

Table 2.2: DACI integration APIs

Phases	APIs	Description
Reservation	*reserve(tenantId, res_desc)*	Generate provider's policies for given cloud resource description
Deployment	*deploy(tenantId)*	Transform policies into contexts and store to the context DB.
Decommission	*release(tenantId)*	Remove tenant's policies and contexts.

2.9.2 High Performance PDP for Tenant Policies

Tenant's policies are isolatedly stored in the *Tenant Authz Policy DB*. Upon receiving a request from *Context Handler*, the PDP service loads equivalent tenant's policies for evaluation. To gain high performance throughput, we use SNE-XACML engine [85] to transform regular XACML policies into X-MIDD data structure with much improved throughput compared to other PDP engines.

We create a pool of PDP instances, each for a tenant policy root. By this way, our design is scalable if we plan to extend PDP in different machines.

2.9.3 Context Resolution and Token Exchange

The Context resolution service implements the multi-tenant access control model by extracting delegation policies of providers and tenants to contexts and storing them in the *Context DB*. It finds the trust context for a given request from *Context Handler*. If the trust context has its issuer at a different domain (a PIP), it can either do one of following:

- Proxy method: The VIP creates a VR proxy to send commands to PIP. It's transparent to end-users. This method is implemented in LICL testbeds. This is the direct approach and acceptable with low control and management traffics because they are centralized and routed via VIP to different PIP domains.

- Push method: The VIP creates a grant-token and relays via end-users to send commands to PIPs as in Section 3.3.2. In general Intercloud services, when the control and management traffics from users to underlying providers are high, this approach is more scalable.

2.9.4 Tenant Policy Administration

A tenant can define its end-users authorization policies as well as inter-tenant delegation policies via policy administration APIs as in Table 2.3.

Table 2.3: Tenant policy administration APIs

Type	APIs	Description
Intra-tenant	*addPolicy(policyId, p)*	Add new policy to the tenant's store.
	updatePolicy(policyId, p)	Update an existing policy
	deletePolicy(policyId)	Delete an existing policy
Inter-tenant	*setTrust(trustee, p)*	Set a new trust relationship between current tenant and *trustee*.
	updateTrust(trustee, p)	Update an existing relationship between current tenant and *trustee*.
	removeTrust(trustee, p)	Remove an existing relationship between current tenant and *trustee*.

Whenever tenants want to add or update policies, the grant constraint in Section 2.5 is checked to make sure no violation happens. Thus, it reduces the authorization overhead by limiting inconsistent decisions.

Table 2.4: VI Datasets

#VI	#Prov. rules	#Inter-tenant rules	#Intra-tenant rules	Total rules
100	401	394	501	1296
300	1217	394	1517	3128
500	2001	500	2501	5002
800	3211	800	4011	8022
1000	3955	1000	4955	9910

2.10 Implementation and Evaluation

2.10.1 Implementation Overview

We develop DACI components as OSGi bundles on Java 1.7. The public DACI interfaces are REST web services based on JAX-RS APIs of the Apache CXF. In our testbed, components in a DACI instance are deployed on the Apache ServiceMix environment [86]. Policies for tenants and providers are stored in a Redis key-value database system [87] with separated key identifiers for each tenant. It guarantees the isolation of policy management among tenants.

The policy generator module uses Jena OWL engine [88] to parse input VI descriptions in INDL [79] and the attribute-based policy template to generate XACML policies as described in Section 2.7.2. They are stored as provider's policies for equivalent VIs.

We use SNE-XACML engine [47] as the core PDP to evaluate intra-tenant policies in the *AuthzService* component. For inter-tenant policies and provider's policies, the *ContextService* component transforms into the context objects, each is composed from a MIDD data structure and the policy's issuer attributes. These context objects are stored persistently and used for context validation purposes.

In our testbed, we build a VM representing the VIP role and two VMs for two different PIPs. Each DACI instance in a PIP's VM is registered with the VIP's DACI instance as a tenant. For testing purpose, we generate sample VI datasets with different VI sizes as in Table 2.4, each VI is one of the following types:

- Type 1: a storage component connected to a VM via a virtual triangular topology network of three virtual routers.
- Type 2: two storage components having network links to a VM.
- Type 3: two VMs having network links to a share storage component.

These VI descriptions are used to generate provider delegation XACML policies. We simulate the inter-tenant operations by generating inter-tenant policies to share resources between tenants, i.e. tenant t_i shares a resource to tenant t_{i+1}. For intra-tenant policies for each tenant, we generate the default policy indicating that the subjects belong to *admin* group of the tenant having all permissions. In practical, inter-tenant and intra-tenant policies are managed by the tenant via the *TenantAdmin* interface.

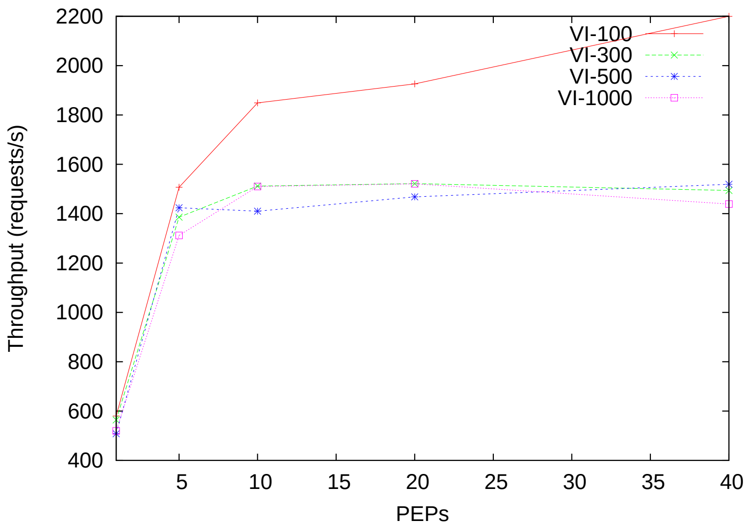


Figure 2.9: Single cloud provider performance evaluation

The DACI for a provider is deployed in a VM with two virtual cores and 4096MB RAM. It runs a ServiceMix instance for DACI and a local Redis server for storing policies.

We use different numbers of PEPs sending requests to the DACI server via the AuthzSvc RESTful interface. They run simultaneously on different machines from the DACI VM, each sends 100 random requests. The execution times of PEPs are measured to calculate the average value.

2.10.2 Evaluation Results

Figure 2.9 shows the performance result for the single provider scenario, where the AuthzService on DACI performs authorization evaluation on provider local resources and does not issue tokens. We observe that throughputs are affected by the number of managed VIs differently. The throughput in VI-100 scenario is higher 12%-40% compared to other scenarios. For scenarios VI-300, VI-500 and VI-1000, with the same number of PEPs, their throughputs are stable. It means that our prototype is scalable for number of resources.

In other aspect, the result also shows that using high number of PEPs for a given dataset in VI-300, VI-500 or VI-1000 can generate enough requests to saturate the AuthzSvc message queue. We measure the AuthzSvc service can handle from 1400 to 1600 requests/s. In our prototype, we intentionally do not apply popular enterprise service patterns like distributed load balancing or decision caching, which can improve system performance more.

2.11 Conclusions

In this chapter, we presented a multi-tenant attribute-based access control model for cloud services in which the access control model is integrated with the cloud infrastructure information description model. Our approach not only can generate provider delegation policy automatically from cloud resource descriptions but also can support multiple levels of delegations with high flexibility for inter-tenant collaborations. Constraints were defined to guarantee consistent and correctness of semantic policy management. We utilized decision diagram mechanisms to attribute-based policy evaluation, which also facilitated the implementation of the proposed context in our model. The prototype was developed, tested and integrated into the GEYSERS project. The evaluation results demonstrated that our prototype has good performance in term of number of cloud resources, clients and policies.

However, the current approach needs to be extended for distributed environments with multiple cloud providers, in which a tenant can play as the provider role by giving its own cloud services. It requires inter-connections between multiple MT-ABAC systems. Chapter 3 will identify and propose mechanisms to solve this problem.

Chapter 3

Multi-tenant Access Control for Intercloud

This chapter is based on the following publications:

- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures," in Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 343–349 [49].
- C. Ngo, Y. Demchenko, and C. de Laat, “Toward a Dynamic Trust Establishment Approach for Multi-provider Intercloud Environment," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012 [89].
- C. Ngo, Y. Demchenko, and C. de Laat, “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services," in Journal of Information Security and Applications (*accepted 2015*) [50].

3.1 Introduction

NIST’s cloud computing reference architectures [5] provide a basis for cooperation between providers to bring integrated cloud services to customers, defined as and referred thereafter as Intercloud [90, 91]. In the general Intercloud architecture, collaborations between providers form a hierarchical multi-level stack of cloud services where each service can compose from lower-level services and also be integrated into upper level ones. In Figure 3.1, IaaS cloud providers can aggregate individual VRs from different PIPs to build up VIs consisting of virtual computing nodes, virtual storage, reserved network links [10]. PaaS and SaaS providers can utilize the outsourced IaaS services to build up their systems. In turn, the end-user work flow systems may be composed from a set of different cloud resources, which requires interactions between cloud providers, even they do not have direct subscription contracts defined by Service Level Agreements (SLAs).

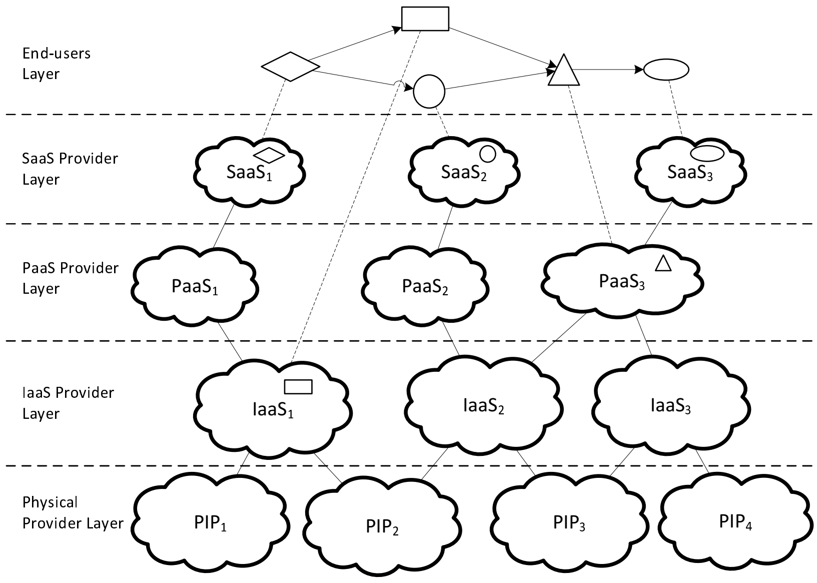


Figure 3.1: An Intercloud scenario

Such cloud collaborations bring challenges on distrusted authorization across multi-domains between providers. The cloud resources while are directly or indirectly managed at a provider domain, can be accessed by unknown entities based on the cloud resource owners consents. In Figure 3.1, the software running on $PaaS_3$ can access to the storage service managed by $IaaS_1$, but located at PIP_1 site. Most current authorization frameworks rely on known identifies of entities or using federated identity management systems with setting up manually when a member want to join to the federation, while the Intercloud requires the dynamic relationship establishment at runtime via third-parties. When collecting distributed decisions, the inconsistency of policies at different domains may lead to high rejected cases, i.e., in a chain of decisions, if the final result is denied, for efficiency, it'd better to be decided at the local domain rather than at a remote one.

We propose a token-based exchanging approach between providers combining with the attribute-based multi-tenant access control model that guarantees the harmonization of distributed authorization policies, thus reduces the denied decision rate to have low system overhead.

3.2 Problem Statement

The model in chapter 2 solves multi-tenant access control problems in a single security domain with a cloud provider. For Intercloud scenarios, a provider could play as a tenant of another provider to utilize its cloud resources. This paradigm can be illustrated in [10] and [49] where the VIP can collect VRs from set of PIPs to compose the VI. Our model in chapter 2 can be extended to support Intercloud scenarios by arrange in hierarchy as follows:

- There are set of cloud providers: $p_i \in P$, each of them runs the model in chapter 2.
- When a provider p_a subscribes cloud resources from set of providers $p_b = \{p_{b_1}, p_{b_2}, \dots p_{b_k}\}$, p_a becomes the tenant of p_{b_i}, thus can manage these resources with its own policies.
- The p_a can also transfer permissions to its tenants. Permissions either targets to local p_a resources, or the remote resources at a provider $p_{b_i} \in p_b$.
- Any p_{b_i} may be a tenant of other providers, so the chain of providers can be extended.

In such scenarios, we need to solve the challenge of distributed authorization in multiple domains. A request from p_a domain may need to be authorized at two domains, first at p_a domain with relevant p_a tenants' policies, then at p_{b_i} domain with the policy issued by p_{b_i} to p_a. The possible approaches to synchronize and collect decisions are either exchanging tokens or exposing policies between domains. The exchanging token approach needs to deal with token management issues, including storing, synchronization, revocation and overhead of using tokens [68]. In Intercloud paradigm, exchanging policies approach may disclose tenants' SLAs out of the provider's domain while still has similar issues with token management [68]. This section proposes a token mechanism that solves token management problems with low overhead on the system performance.

3.3 Extended Model for Multiple Providers

3.3.1 Constraints in Distributed Authorizations

The potential problem in distribute authorization is the conflicting decisions between domains, resulting to the high rejection requests rate at remote domains and increasing system overhead. Preferably, denied requests should be answered as soon as possible at their local domains, rather than at a remote domain in the chain of distributed authorization. It can be solved by establishing grant constraints between the policies at tenant side p_a with policies at the p_b provider sides.

In the above scenario, the provider p_{b_i} issues a policy with the context c_{a_i} for the provider p_a. At the p_a, instead of contexts created by p_a, contexts $c_{a_i|i=\overline{1 \dots k}}$ become the root context. All contexts created by p_a must be confined by the root contexts. It can be excepted for local resources X_r physically owned by p_a. To synchronize contexts c_{a_i} between p_{b_i} and p_a, we base on the SLA describing subscribing resources between them. According to the information model [79], the SLA request is described by INDL semantic concepts and synchronized upon provisioning and re-planning. We then can use SPARQL and policy generation techniques to extract constrained contexts and update to the trusted root list, which is similar to policy generation in Section 2.5.

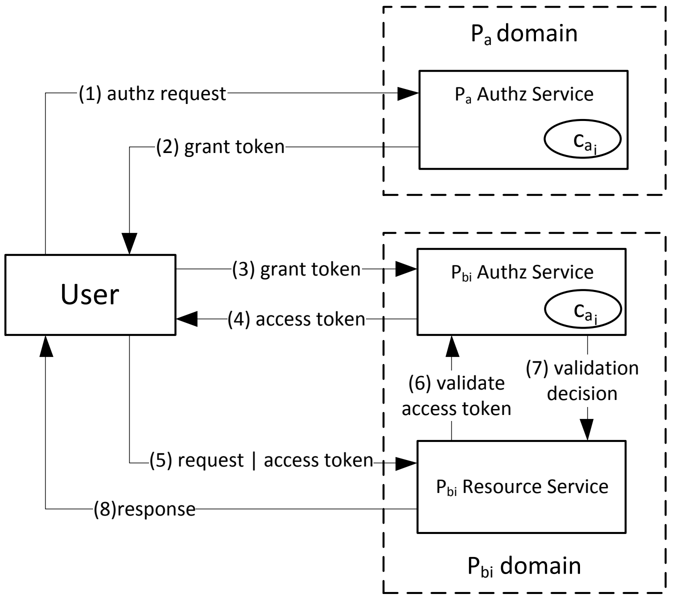


Figure 3.2: Exchanging tokens in Intercloud: grant token and access token

3.3.2 Token Exchange in Intercloud

The distributed authorization workflow can be the push sequence as in Figure 3.2. It requires that the user needs to have an access token to verify it's allowed to access the resources at remote provider p_{b_i}'s domain. The grant token is initially issued by the first provider in the chain p_a as the consent by p_a to subsequent provider p_{b_i}. The p_{b_i} must validate the token issuer p_a, then evaluate the request attribute embedded inside the token against its policies. If the decision is positive and the target resource is located in its local domain, p_{b_i} issues an access token allowing the user to access it. Otherwise, if the target resource is located at another domain, p_b issues another grant token to user for further distributed authorization process. In this sequence, the communication between authorization services at providers is relayed through the user via exchanging grant tokens and access token.

3.3.2.1 Grant Token

The grant token needs to have the following information:

- Request content approved by the issuer, who allows the request to act on behalf of the issuer: it usually is the vector of attributes including issuer's subject attributes.
- The approval proof of the issuer: this proof can be enforced by the digital signature mechanism of the issuer, either based on a digital signature using public cryptography or a message authentication code algorithm using symmetric cryptography.

- The lifetime limitation.
- The proof-of-procession of the user, so the issued access token is not a bearer token and only targets for the user. It's either the user's public key, or the session shared secret key generated by the user.

For the public key cryptography approach, we propose the grant token issued by p_a and returned to the user u as follows:

$$\begin{aligned} &X := \{X_{p_a}, X_r, X_e\} \\ &m := X|t|pk_u \\ &granttoken := SK(sk_{p_a}, m) \end{aligned} \tag{3.1}$$

with $SK(sk_{p_a}, m)$ is the annotation that the message m is signed by secret key sk_{p_a} of the provider p_a. The pk_u is the user's public key, t is the lifetime and X is the vector of attribute request containing p_a's attributes. This grant token allows user to request on behalf of p_a to the remote domain at p_b.

For the symmetric key cryptography approach, the grant token has the following information:

$$\begin{aligned} &m := X|t|k_u \\ &hmac := MAC(K_{p_a,p_{b_i}}, m) \\ &ek := E(K_{p_a,p_{b_i}}, k_u) \\ &granttoken := \{X|t|ek|hmac\} \end{aligned} \tag{3.2}$$

with $K_{p_a,p_{b_i}}$ is the shared secret key between the provider p_a and p_{b_i}; MAC is a message authentication code algorithm; k_u is the session key of the user; ek is the encryption of k_u by the $K_{p_a,p_{b_i}}$.

3.3.2.2 Access Token

According to the public key approach in Formula (3.1), the access token issued by p_{b_i} to the user u is constructed as follows:

$$accesstoken := SK(sk_{p_{b_i}}, tid|t) \tag{3.3}$$

with t is the issuing timestamp, tid is the identifier to the cached authorization session stored at p_{b_i}, which contains access token lifetime, user's associated key pk_u and the involved attributes X.

With symmetric key approach in Formula (3.2), the access token contains following:

$$accesstoken := E(k_u, tid|t|stoken) \tag{3.4}$$

$$k_{u,p_{b_i}} = k_u|stoken \tag{3.5}$$

with $stoken$ is the secret generated value by p_{b_i} shared to the user. The consequent requests from the user to p_{b_i} are signed with the session key $k_{u,p_{b_i}}$.

After having the access token, user accesses the protected resource at p_{b_i}. Upon receiving user's request with access token, the p_b's resource service validates the access token with either pk_{b_i} for public key scheme, or $k_{u,p_{b_i}}$ for symmetric key scheme. If comparison between the request with involved attributes X is positive, the service will serve the request.

3.4 Implementation and Evaluation

3.4.1 Implementation Overview

Our exchanging token approach is implemented in the *TokenService* of the DACI. The service has a public/private key-pair used for issuing and validating tokens. Upon registration, each tenant is bound with a separate public/private key-pair used for Intercloud communication scenario as described in Section 3.1. In our key management implementation, we choose the RSA algorithm with 2048 bits key length. For digital signature used in issuing tokens, we define the token structure in XML schema and use the XML digital signature standard [92] implemented in the Apache XML security library [93]. We choose RSASSA-PKCS1-v1.5 signature scheme with SHA-1 algorithm [94]. DACI uses Bouncy Castle v1.49 [95] as the Java cryptographic provider.

We deploy DACI instances with TokenService in separate VMs having two virtual cores and 4096 MB RAM. Each VM represents a cloud provider running DACI with the sample datasets in Table 2.4 as in Chapter 2.

In our inter-provider test scenarios, we have two DACIs for P_a and P_b providers, in which P_a subscribes resources of the P_b as described in Section 3.2. PEPs at the user side of the P_a send requests to access to the resource at P_b, so DACI of the P_a needs to evaluate its local policies prior issuing grant-tokens for further authorization at P_b. Compared to the intra-provider scenario in the last chapter, the token issues and validations increase overhead of the original DACI system.

3.4.2 Evaluation Results

From our experiments, the performance tests show that on average, the response time for an authorization request with issuing grant-token is 320 ms, which is significantly slower than the response time in the intra-provider scenario. The overhead here mostly comes from the digital signing tokens with RSA 2048 bits key-length for every issued token and the XML messages serialization/deserialization. Therefore, we are developing a hybrid key management scheme in which tenants and providers use shared secret keys in communications, which are established and refreshed periodically based on the public/private key-pairs. The symmetric key scheme using message authentication code could improve the system performance significantly compared to the public key scheme.

3.5 Conclusions

In this chapter, we extend the MT-ABAC for distributed, multiple collaborative cloud providers in hierarchy to support Intercloud scenarios with exchanging tokens approach. In future work, we will improve key management model for Intercloud using combining public-key and symmetric cryptography, which could improve the system performance in the Intercloud communications using tokens. We are planning to develop adapter layers between our DACI system using INDL with popular cloud management systems like OpenStack, CloudStack or Eucalyptus, thus could integrate the DACI with these systems. Regarding authorization policy language, beside XACML in XML profile, we plan to support others as well as supporting our DACI with legacy on-premise authorization systems.

Chapter 4

Logical Model and Mechanisms for XACML

This chapter is based on the following publications:

- C. Ngo, M. X. Makkes, Y. Demchenko, and C. de Laat, “Multi-data-types interval decision diagrams for XACML evaluation engine," in Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on, 2013, pp. 257–266 [47].
- C. Ngo, Y. Demchenko, and C. de Laat, “Decision Diagrams for XACML Policy Evaluation and Management," In Computers & Security 49 (2015), pp. 1–16 [83].

4.1 Introduction

XACML is an authorization policy language in XML format based on the ABAC model. It composes policies from set of attribute criteria joined by logical operators to decide if authorization requests are granted. XACML is scalable in arranging policies in the hierarchical order in the repository. The policy language also supports delegations, obligations and advices, that makes it applicable in many areas such as networking, grids, clouds, enterprise organization and management. However, expansions of policies to address system scales will increase the complexity of the repository, which drops the policies evaluation performance.

XACML policies has complex structures containing a sophisticated logical model as follows:

- Policies are organized hierarchically in a policy-tree with rules, policies and policy-sets elements. The tree contains internal nodes and external nodes. An internal node can either be a policyset or a policy. Children of a policyset node can be other policysets or policies. Children of a policy are rules, which are external nodes. Because children can produce conflicting decisions, parent nodes can resolve them by predefined combining algorithms.

- Policy decisions are not only *permit* and *deny*, but also other intermediate values to handle error and un-matched situations such as *not-applicable*, *indeterminate* decisions (see section 4.3). It means that operations on combining policies' decisions cannot be derived from binary logical operators. They should be defined in multi-valued logical domains.

- Not all attributes are processed equally, some of them are marked as critical (with the flag *"MustBePresent=true"*): during the evaluation, the missing of these attributes should yield *indeterminate* values rather than the *not-applicable*.

- Because policies have their own predicates to match with requests, attribute comparisons are scattered in the policy-tree. Thus, typical implementations discussed in [96] often have redundancies in evaluations: an attribute may be compared multiple times in different policy nodes.

With these characteristics, there are challenges to propose high performance policy evaluation solutions or resolve policy analysis and management problems. We need practical mechanisms that not only can gather predicates and efficiently reduce them in aware of combining algorithms, but also guarantee multi-valued logical semantics of the XACML.

To facilitate the high performance policy evaluation mechanism in the access control systems for clouds using XACML [48, 49, 97], from recent policy evaluation approaches [98, 99], and state-of-the-art of XACML engines performance [96], in this chapter we analyze the logic behind XACML standard and propose a practical decision diagram mechanism. It includes interval partition processing, MIDDs and their combination algorithms, which are then applied to design a high performance policy evaluation engine in Chapter 5. Our contributions in this chapter are as follows:

- Analyze the logic of XACML components evaluation, which essentially is a many-valued logic system with equivalent operators on different domains.

- Define the MIDD and X-MIDD data structures definitions representing logical expressions in XACML. Along with them, we define interval partition processing and related operators, which are then used to process XACML elements.

- Compared to related work, our approach covers most of XACML features in XACML 3.0 [36], including continuous data-types, complex comparisons, correctness of combining algorithm semantics, error handling and critical attribute setting.

The proposed mechanisms can also be applied to solve XACML policy management problems such as policy comparison, policy redundancy detection, policy testings or authorization reverse queries.

The rest of the chapter is organized as follows. Section 4.2 reviews the related work on policy analysis, management, integration and high performance evaluation.

Section 4.3 analyzes XACML logic that provides the basis for the proposed solution. Section 4.4 formulates the approach to evaluate the complete logical expressions using interval decision diagrams. Section 4.5 defines fundamental operations to process intervals, partitions and decision diagrams. These materials and mechanisms in this chapter can be applied to solve different policy management problems, which are pointed out in Section 4.6. Finally, Section 4.7 concludes the chapter.

4.2 Related Work

There are numerous prior works on access control policies that mainly focus on policy verification, analysis and testing to detect and remove redundancy [100–103]. Authors in [100] used propositional logic in XACML to identify properties of given policies and analyze the change-impact of two policies to summarize their differences. The proposal was implemented in the Margrave project using Multi-Terminal Binary Decision Diagram (MTBDD) [104] as the underlying mechanism. Because of using one binary variable for each attribute-value pair, their approach is only applicable for policies containing all predefined attribute values. In other aspect, by modeling decisions as binary values, it omitted error use-cases handling in all XACML evaluation semantics. Li & Tripunitara [101] and Hu & Ahn [103] proposed methodologies to verify and correct policies under the RBAC model. Kolovski et al. [102] used description logic to represent XACML policies and use DL reasoners for analysis tasks such as policy comparisons, verification and querying. However, because description logic could only covers a subset of XACML, this approach did not handle complex comparisons, *indeterminate* decisions handling as well as left out *one-applicable* combining algorithm. Masi et al. [105] formalized the XACML 2.0 semantics and proposed an alternative syntax supporting policy composition. They implemented a tool to compile policies into Java classes following the proposed semantic rules, where these classed are executed to compute policy decisions.

Policy integration and composition was introduced firstly by Bonatti et al. [106]. They defined an algebra with constraints to compose and translate policies into logic programs. However, the algebra did not bind with any practical policy language. Mazzoleni et al. [107] proposed the policy integration preferences, which is an XACML extension that specified how to integrate policies from different parties. In spite of that, they did not show any applicable mechanisms for such integration. Bruns et al. [108] attempted to use Belnap logic to formalize XACML 2.0, in which they map four logic values to XACML policy decisions. Even that, the logic of XACML is different from Belnap logic because the *indeterminate* values cannot map to any Belnap logical value. Subsequently, [109] used $\mathcal{D}$-algebra to formulate combining algorithms in XACML 2.0. But the $\mathcal{D}$-algebra did not correctly represent *indeterminate* decisions: e.g., with permit-override algorithm for *indeterminate-p* ($\{p, \frac{n}{a}\}$) and *deny* ($\{d\}$), the combination should be *indeterminate-dp* ($\{p, d, \frac{n}{a}\}$) rather than the $\{p, d\}$. Rao et al. [110] defined a 3-value Fine-grained Integration Algebra (FIA) which attempted to formulate operations of XACML elements via FIA operators. They used MTBDD to represent their approach. However, the FIA could not represent all *indeterminate* values and did not distinguish differences between *target* and *rule* evaluations which operate on different domains.

To solve the problem of policy evaluation performance, in the preliminary version [111] and later [98], Liu et al. attempted to transform policies into decision diagrams. Their approach started with the numericalization by mapping all attribute values into integer numbers, which was only applicable for equally comparisons policies with predefined attribute values. Although boosting the evaluation performance, this proposal only covered a subset of XACML policies when did not support the complex attribute comparisons, correct *indeterminate* decisions handling, critical attribute evaluation as well as obligations. Marouf et al. [112] reordered most frequent applicable policies using clustering techniques based on statistics from past requests, which could increase the chance in evaluating only a subset rather than the whole policies. This technique can only improve performance when incoming requests repeated with high probability, rather than the uniform random requests. Moreover, reordering policies would not support obligations handling.

The approach of Pina Ros et al.[99] extended [111] with interval techniques. They kept original data types of attributes with more supported comparison operators. The approach used two trees: the Matching Tree (MT) is a decision diagram built up from extracted predicates in target expressions, and the Combining Tree (CT) is at each MT's leaf nodes. A CT contains a subset of the policy tree with only applicable rules or policies without *Target* elements. The CT evaluation followed defined combining algorithms in the subtree. This approach is different from [111] when it stores the subset of policy tree rather than the flat of applicable rules. However, this proposal has following flaws: first, it ignored critical attribute evaluation handling: e.g., if an attribute is missed from the request, the evaluation will yield a *indeterminate* with a critical attribute predicate rather than *not-applicable*. This leads to different decisions when combining with other rules, such as with deny-override algorithm for *indeterminate* and *deny*, the outcome will be $indeterminate_{DP}$, rather than combining *not-applicable* and *deny* to *deny*. Second, because the CT only contains applicable rules equivalent to the matching path from the root of the MT, this approach could not handle error well if an attribute in the path is missed from the request. The evaluation would be blocked without giving any decision, while in practice, it always has a consistent answer for a given request.

Ramli et al. [113] presented the most recent work analyzing the logic behind XACML, in which evaluation semantics relied on operators over domains V_3 and V_6. Comparing to previous work, it covered most aspects in analyzing XACML logic, however, this approach still omitted to handle critical attribute setting, obligations as well as did not have any applicable implementation.

4.3 Semantics of XACML Policy Components

4.3.1 Abstraction

XACML elements [36] are organized in a hierarchical order, which contains policysets, policies and rules. Each of them has a *Target* expression as the criteria for incoming requests. The returned decision is either defined in the rule's *"effect"* property, or combined decisions of children rules, policies or policysets.

Attributes in XACML have different data types. Without loss of generality, we assume a XACML attribute domain D_i can be normalized to either $\mathbb{R}$ or a sub-domain of $\mathbb{R}$. So we can say that D_i is the totally ordered domain representing a continuous data type.

A XACML request $X = \{x_1, x_2, \dots x_n\}$ is the set of attribute values, each item $x_i \in D_i$ with D_i is an XACML attribute domain.

XACML specification [36] describes elements in XML using XML Schema Definition (XSD). For short representation purpose, we abstract main XACML elements that our logical analysis focuses in Table 4.1 using Backus-Naur Form notation.

The *<object>* in obligation and advice represents a general object. The *<attr-id>* identifies an attribute $a_i \in D_i$, the *<attr-value>* specifies a constant value $v_i \in D_i$.

The combining algorithms are in the Table. 4.2. They define how to combine children's decisions to the policy or policyset result.

Evaluation values of Match, AllOf, AnyOf, Target and Condition elements are summarized in Table 4.3, while the decision values of Rule, Policy and Policyset elements are in Table 4.4. XACML extends decision values with *"not-applicable"* and *"indeterminate"*, compared to previous policy languages with only *"permit"* an *"deny"* decisions. The *"not-applicable"* means that the request does not match with the rules or policies, while the *"indeterminate"* values indicate that some errors may occur during evaluation (e.g., requests miss the critical attribute, errors in parsing policies). Because of this feature, XACML essentially is as a many-valued logic policy language, while prior work did not analyze and solve following this direction [98–100, 111].

A sample XACML policy is shown in the Listing 4.1. Originally it is the XML documents following XACML schema standard [36]. However in this example, we illustrate policies and rules as JSON objects for short representation. Target elements in the example are expressed as logical expressions.

In the sample policy, the *'vol'* is the *volume* attribute, *'p'* is the *price* attribute and *'t'* is the *time* attribute. In the rule R_0, *'vol'* attribute is marked as critical by the underline, otherwise it is optional.

4.3.2 Predicate Elements

4.3.2.1 Match element

The XACML Match element is composed from a tuple of *(match-id, v, x)* where the *match-id* is a two-operand predicate Boolean function, *v* is the attribute value as the first operand and x is the attribute-id as the second one.

The match evaluation returns one of values in Table 4.3 as follows:

- If the comparison returns *true*, the result is *"Matched"*.
- If there's either no attribute x found in the request, or the comparison is *false*, the result is *"No-matched"*.

Table 4.1: XACML abstract syntax

$\langle$*Policyset*$\rangle$ ::= ($\langle$*target*$\rangle$, $\langle$*Policy-list*$\rangle$, $\langle$*combine-algo*$\rangle$)

$\langle$*Policy-list*$\rangle$ ::= $\langle$*Policy-item*$\rangle$ | $\langle$*Policy-item*$\rangle$ $\langle$*Policy-list*$\rangle$

$\langle$*Policy-item*$\rangle$ ::= $\langle$*Policy-set*$\rangle$ | $\langle$*Policy*$\rangle$

$\langle$*Policy*$\rangle$ ::= ($\langle$*target*$\rangle$, $\langle$*Rule-list*$\rangle$, $\langle$*combine-algo*$\rangle$)

$\langle$*Rule-list*$\rangle$ ::= $\langle$*Rule*$\rangle$ | $\langle$*Rule*$\rangle$ $\langle$*Rule-list*$\rangle$

$\langle$*Rule*$\rangle$ ::= ($\langle$*target*$\rangle$, $\langle$*condition*$\rangle$, $\langle$*effect*$\rangle$, $\langle$*obligation-exprs*$\rangle$, $\langle$*advice-exprs*$\rangle$)

$\langle$*target*$\rangle$ ::= $\langle$*anyof-list*$\rangle$

$\langle$*anyof-list*$\rangle$ ::= | $\langle$*anyof*$\rangle$ $\langle$*anyof-list*$\rangle$

$\langle$*anyof*$\rangle$ ::= $\langle$*alloff-list*$\rangle$

$\langle$*allof-list*$\rangle$ ::= $\langle$*allof*$\rangle$ | $\langle$*allof*$\rangle$ $\langle$*allof-list*$\rangle$

$\langle$*allof*$\rangle$ ::= $\langle$*match-list*$\rangle$

$\langle$*match-list*$\rangle$ ::= $\langle$*match*$\rangle$ | $\langle$*match*$\rangle$ $\langle$*match-list*$\rangle$

$\langle$*match*$\rangle$ ::= ($\langle$*match-id*$\rangle$, $\langle$*attr-value*$\rangle$, $\langle$*attr-id*$\rangle$)

$\langle$*obligation-exprs*$\rangle$ ::= | $\langle$*obligation-expr*$\rangle$ $\langle$*obligation-exprs*$\rangle$

$\langle$*obligation-expr*$\rangle$ ::= ($\langle$*object*$\rangle$, $\langle$*fulfill-on*$\rangle$)

$\langle$*fulfill-on*$\rangle$ ::= $\langle$*effect*$\rangle$

$\langle$*advice-exprs*$\rangle$ ::= | $\langle$*advice-expr*$\rangle$ $\langle$*advice-exprs*$\rangle$

$\langle$*advice-expr*$\rangle$ ::= ($\langle$*object*$\rangle$, $\langle$*applies-to*$\rangle$)

$\langle$*applies-to*$\rangle$ ::= $\langle$*effect*$\rangle$

$\langle$*match-id*$\rangle$ ::= eq | ne | gt | lt | ge | le

$\langle$*combine-algo*$\rangle$ ::= po | do | fa | ooa | pud | dup

$\langle$*effect*$\rangle$ ::= ‘‘`permit`” | ‘‘`deny`”

$\langle$*request*$\rangle$ ::= $\langle$*attribute-list*$\rangle$

$\langle$*attribute-list*$\rangle$ ::= $\langle$*attribute*$\rangle$ | $\langle$*attribute*$\rangle$ $\langle$*attribute-list*$\rangle$

$\langle$*attribute*$\rangle$ ::= ($\langle$*attr-id*$\rangle$, $\langle$*attr-value*$\rangle$)

Table 4.2: XACML combining algorithms

Combining algorithms	**Annotations**
Permit-override	po
Deny-override	do
First-applicable	fa
Only-one-applicable	ooa
Permit-unless-deny	pud
Deny-unless-permit	dup

Table 4.3: XACML evaluation values for elements: Match, AllOf, AnyOf, Target and Condition

Evaluation values	**Annotations**
Matched	T
No-matched	F
Indeterminate	IN

Table 4.4: XACML decision values for Rule, Policy and Policyset elements

Decision values	**Annotations**
Permit	P
Deny	D
NotApplicable	N
Indeterminate{P}	IN_P
Indeterminate{D}	IN_D
Indeterminate{PD}	IN_{PD}

- If there's no attribute x found in the request, and this attribute is marked as critical with the flag *MustBePresent=true*, the result is *"indeterminate*", meaning that an error occurs.

Denoting the set $V_M := \{T, F, IN\}$, the match evaluation can be represented as the function mapping from an attribute domain D_i to match values V_M:

$$\mu(x_i) : D_i \to V_M \tag{4.1}$$

4.3.2.2 AllOf, AnyOf and Target elements

The AllOf element is a list of Match items joined by the $\wedge$ operator. Because the Match items return values in V_M domain, the $\wedge$ operator is extended from the regular "AND" boolean operator:

$$\prod_{i=1}^{k} m_i := m_1 \wedge m_2 \cdots \wedge m_k = \begin{cases} T & \text{if } \forall i \in [1,k], m_i = T \\ F & \text{if } \exists i \in [1,k], m_i = F \\ IN & \text{if } \forall i \in [1,k], m_i \neq F; \exists j \in [1,k], m_j = IN \end{cases} \tag{4.2}$$

The AnyOf element is a list of AllOf items joined by the $\vee$ operator, which is

Listing 4.1: Sample XACML policies

```
Policy $P_0$: { combine−algo: po,
     target : $(vol \geq 100) \wedge (vol \leq 500)$,
     rule−list: [$R_1, R_2$],
}
Rule $R_1$: { effect: $permit$,
     target: $[(100 \leq \underline{vol} \leq 150) \wedge (12 \leq t \leq 17) \wedge (3 \leq p \leq 4)] \vee$
       $[(300 \leq vol \leq 500) \wedge (1 \leq p \leq 2)] \vee$
       $[[(100 \leq vol \leq 150) \wedge (6 \leq t \leq 9) \wedge (1 \leq p \leq 2)]$,
     obligations: [$\{O_1, permit\}$]
}
Rule $R_2$: { effect: $deny$,
     target: $[(vol = 100) \wedge (t = 17)] \vee [(100 \leq vol \leq 300) \wedge (t = 9)] \vee [(vol = 500) \wedge (t \geq 12)]$,
     obligations: [$\{O_2, deny\}$]
}
```

defined as:

$$\coprod_{i=1}^{k} a_i := a_1 \vee a_2 \cdots \vee a_k = \begin{cases} T & \text{if } \exists i \in [1,k], a_i = T \\ F & \text{if } \forall i \in [1,k], a_i = F \\ IN & \forall i \in [1,k], a_i \neq T, \exists j \in [1,k], a_j = IN \end{cases} \tag{4.3}$$

The Target element joins AnyOf elements by the $\wedge$ operator like in Eq. (4.2). In other aspect, the Target evaluation over incoming request $X \in D_1 \times D_2 \times \ldots D_n$ is also defined as the function τ:

$$\tau(X) : D_1 \times D_2 \times \ldots D_n \to V_M \tag{4.4}$$

The XACML defines that an empty Target element returns the T value.

From Eq. (4.2) and Eq. (4.3), we can see that these operators along with the set V_M form a lattice $(V_M, \leq)$ with the order $F \leq IN \leq T$; $\wedge$ is the *meet* operator; $\vee$ is the *join* operator.

4.3.2.3 Condition element

The Condition element represents a complex logical expression evaluated by the set of attributes X in the request to return a value in the V_M domain. Without loss of generality, we can denote the Condition element as the function κ:

$$\kappa(X) : D_1 \times D_2 \times \ldots D_n \to V_M \tag{4.5}$$

4.3.3 Rules and Policies

4.3.3.1 Rule Evaluation

The rule $R = \{t, c, e\}$ in which t, c, e are Target, Condition and Effect elements, respectively, is evaluated against a request X. The decision is based on the combination of Target and Condition results, along with the effect value as in Table 4.5.

Table 4.5: XACML rule evaluation specification

Target	Condition	Rule Value
T	T	Effect e
T	F	N
T	IN	IN_P if $e = P$, IN_D if $e = D$
F	any value	N
IN	any value	IN_P if $e = P$, IN_D if $e = D$

Denoting the set $E := \{P, D\}$ containing effect values and the set $V_R := \{P, D, N, IN_P, IN_D, IN_{DP}\}$ having decision values from Table 4.4, the rule evaluation can be represented as follows:

$$R(t, c, e) : V_M \times V_M \times E \to V_R \tag{4.6}$$

in which $t, c \in V_M$ are the results of the Target and the Condition evaluations, respectively; $e \in E$ is the Effect value. According to Table 4.5, the function $R(t, c, e)$ is evaluated as:

$$R(t, c, e) = \begin{cases} P & \text{if } t \wedge c = T \text{ and } e = P \\ D & \text{if } t \wedge c = T \text{ and } e = D \\ N & \text{if } t \wedge c = F \\ IN_e & \text{otherwise} \end{cases} \tag{4.7}$$

The denotation IN_e means the value IN_P if $e = P$ and IN_D if $e = D$.

4.3.3.2 Policy and Policyset Evaluations

Policy and Policyset evaluations are similar. Their decisions are relied on the Target element and the combined decision of their children using a combining algorithm. According to the XACML 3.0 standard, their evaluations are summarized in Table 4.6.

Table 4.6: XACML Policy/Policyset evaluation specification

Target	Combining-algo decisions	Policy/Policyset decisions
T	any value	specified by the combining algorithm
F	any value	N
IN	N	N
	P	IN_P
	D	IN_D
	IN_{DP}	IN_{DP}
	IN_P	IN_P
	IN_D	IN_D

Denoting a policy $P = \{t, ca, \{R_i\}_{i=1}^{k}\}$ in which t is the evaluation result of the policy's target: $t = \tau(X) \in V_M$; ca is a combining algorithm in Table 4.2 and R_i is a child rule of the policy. The policy evaluation P is represented as the function:

$$P(t, \omega_{ca}(R_i)_{i=1}^{k}) : V_M \times V_R \to V_R \tag{4.8}$$

which $\omega_{ca}(R_i)_{i=1}^{k} \in V_R$ is the combining function of children decisions in Section 4.3.4.

According to Table 4.6, denoting $\psi = \omega_{ca}(R_i)_{i=1}^{k}$, the function in Eq. (4.8) is evaluated as:

$$P(t, \psi) := t \,\overline{\wedge}\, \psi = \begin{cases} \psi & \text{if } t = T \\ N & \text{if } t = F \text{ or } t = IN, \psi = N \\ IN_P & \text{if } t = IN \text{ and } \psi \in \{P, IN_P\} \\ IN_D & \text{if } t = IN \text{ and } \psi \in \{D, IN_D\} \\ IN_{DP} & \text{if } t = IN \text{ and } \psi = IN_{DP} \end{cases} \tag{4.9}$$

Given a policyset $PS = \{t, ca, \{P_i\}_{i=1}^{k}\}$, the evaluation also relies on Eq. (4.9) to combine its policies' decisions.

4.3.4 Combining Algorithms

XACML 3.0 combining algorithms operate on the V_R domain, which are used to form the ancestor's decision according to Eq. (4.9). Denoting a combining operator as the ω_{ca}, in which ca is the identifier of an algorithm in Table 4.2:

$$w_{ca}(v_1, v_2, \ldots v_k) : V_R^k \to V_R \tag{4.10}$$

with $v_i \in V_R$ is the child decision.

Combining functions in Table 4.2 are defined as follows:

$$\omega_{po}(v_1, v_2, \ldots v_k) = \begin{cases} P & \text{if } \exists v_i = P \\ D & \text{if } \forall i \in [1,k], v_i \notin \{IN_P, IN_{DP}\} \text{ and } \exists v_j = D \\ N & \text{if } \forall i \in [1,k], v_i = N \\ IN_P & \text{if } \forall i \in [1,k], v_i \in \{IN_P, N\}, \exists v_j = IN_P \\ IN_D & \text{if } \forall i \in [1,k], v_i \in \{IN_D, N\}, \exists v_j = IN_D \\ IN_{DP} & \text{otherwise} \end{cases} \tag{4.11}$$

$$\omega_{do}(v_1, v_2, \ldots v_k) = \begin{cases} P & \text{if } \forall i \in [1,k], v_i \notin \{IN_D, IN_{DP}\} \text{ and } \exists v_j = P \\ D & \text{if } \exists v_i = D \\ N & \text{if } \forall i \in [1,k], v_i = N \\ IN_P & \text{if } \forall i \in [1,k], v_i \in \{IN_P, N\}, \exists v_j = IN_P \\ IN_D & \text{if } \forall i \in [1,k], v_i \in \{IN_D, N\}, \exists v_j = IN_D \\ IN_{DP} & \text{otherwise} \end{cases} \tag{4.12}$$

$$\omega_{fa}(v_1, v_2, \dots v_k) = \begin{cases} P & \text{if } \exists v_i = P \text{ and } \forall j \in [1, i), v_j \notin \{P, D\} \\ D & \text{if } \exists v_i = D \text{ and } \forall j \in [1, i), v_j \notin \{P, D\} \\ N & \text{if } \forall i \in [1, k], v_i = N \\ IN & \text{if } \forall i, v_i \in \{N, IN\} \text{ and } \exists v_j = IN \end{cases} \quad (4.13)$$

$$\omega_{ooa}(v_1, v_2, \dots v_k) = \begin{cases} P & \text{if } \exists! v_i = P \text{ and } \forall j \neq i, v_j = N \\ D & \text{if } \exists! v_i = D \text{ and } \forall j \neq i, v_j = N \\ N & \text{if } \forall i, v_i = N \\ IN & \text{otherwise} \end{cases} \quad (4.14)$$

$$\omega_{pud}(v_1, v_2, \dots v_k) = \begin{cases} D & \text{if } \exists i \in [1, k], v_i = D \\ P & \text{otherwise} \end{cases} \quad (4.15)$$

$$\omega_{dup}(v_1, v_2, \dots v_k) = \begin{cases} P & \text{if } \exists i \in [1, k], v_i = P \\ D & \text{otherwise} \end{cases} \quad (4.16)$$

We can see that for XACML elements, the *Match*, *AllOf*, *AnyOf* and *Target* elements operate over V_M domain, while the rule, policy and policyset evaluations use operators in V_R domains. In the next section, we define data structures and algorithms representing such elements and related operators, which facilitate the XACML evaluation implementation.

4.4 Multi-data-types Interval Decision Diagrams

4.4.1 Introduction

Binary Decision Diagram (BDD) [114] and its extensions (e.g., Interval Decision Diagram (IDD) [115], Multi-Terminal Interval Decision Diagram (MTIDD) [116]) are popular in model checking, verification, firewall and policy analysis. However, they are not fully suitable in XACML processing. In their approaches, attribute data-types limit in discrete domains [115] or only using equality comparisons [98]. In general, actual attributes use continuous data-types with different comparable operators.

It is also possible to apply BDD techniques by using a variable for each attribute-value pair in the proposal of [100]. However, the depth of decision diagrams will be the product of the number of unique values and the number of attributes, therefore time and space complexities are much higher than our approach.

In this section, we construct a decision diagram based approach to implement operators described in the previous section efficiently. We define MIDD data structures with equivalent operators to meet such requirements. First, the section will revisit the basic logical function decomposition for continuous variables, which is represented by the decision diagram $G(V, E)$. However, to match it with our functions over V_M and V_R domains in section 4.3, we need to extend the basic decision diagram into equivalent MIDD and X-MIDD diagrams. Then, we transform operators in section 4.3 to algorithms over MIDD and X-MIDD as in the next section.

4.4.2 Logical Function Decomposition

Denoting a multi-variable logical function with following signature:

$$f : D_1 \times D_2 \ldots \times D_n \rightarrow \{true, false\} \tag{4.17}$$

where D_i is a totally ordered domain representing a continuous data-type in XACML. Denoting vector $X = (x_i | i = 1..n, x_i \in D_i)$, Eq. (4.17) is also seen as: $f(X) \rightarrow \{true, false\}$

Definition 4.1 (Data interval)**.** A data interval $I \subseteq D_i$ is a range of values in the domain D_i which is formed by two endpoints. It can either be an open, closed or half-closed interval, depending on whether endpoints are included in the interval.

Example: The sample policy (Listing 4.1) has different intervals for the *'vol'* variable: in the rule R_1, $vol \in [100, 150]$ or $vol \in [300, 500]$; or in the policy P_0, $vol \in [100, 500]$.

Definition 4.2 (Interval partition)**.** An interval partition P is a set of disjoint intervals in the domain D_i: $P = \{I | I \subseteq D_i : \forall I_i, I_j \in P, i \neq j, I_i \cap I_j = \emptyset\}$

Example: If we want to represent the *'vol'* variable having values in the range $[100, 150]$ or $[300, 500]$, we define $vol \in \{[100, 150], [300, 500]\}$. It is the interval partition containing disjoint intervals.

Given an interval partition P, the denotation $x_i \in P$ means that $\exists I \in P$, s.t $x_i \in I$.

We define a boolean function $h_{x_i}(P)$ as:

$$h_{x_i}(P) = \begin{cases} 0 & \text{if } x_i \notin P \\ 1 & \text{if } x_i \in P \end{cases} \tag{4.18}$$

Function in Eq. (4.17) is called independent with $x_i \in X$ in the interval partition P when:

$$\forall a, b \in P, f(X|_{x_i:=a}) = f(X|_{x_i:=b}) \tag{4.19}$$

In this case, we denote $f_{x_i^P}$ as the partial function:

$$f_{x_i^P} := f(x_1 ..., x_{i-1}, b, x_{i+1}, .., x_n)|_{\forall b \in P} \tag{4.20}$$

Example: The function f below is said to be independent with vol in the partition $P = \{[100, 150]\}$:

$$f(vol, t, p) = (100 \leq vol \leq 150) \wedge (12 \leq t \leq 17) \wedge (3 \leq p \leq 4)$$

So the partial function $f_{vol^{[100,150]}} = (12 \leq t \leq 17) \wedge (3 \leq p \leq 4)$.

Given a domain D_i, the set of partitions $\mathcal{P}(D_i) = \{P_1, P_2 ..., P_{d_i}\}$ is called to cover the domain D_i when

$$D_i = \bigcup_{P \in \mathcal{P}(D_i)} \left(\bigcup_{I \in P} I \right) \tag{4.21}$$

The cover $\mathcal{P}(D_i)$ is disjoint if there's no common interval between them:

$$\forall i, j \in [1, d_i], i \neq j : P_i \cap P_j = \emptyset \tag{4.22}$$

According to Boole-Shannon expansion, the function f can be decomposed to set of partial functions in respect of variable x_i against a disjoint, covered partition $\mathcal{P}(D_i)$

$$f(X) = \bigvee_{P \in \mathcal{P}(D_i)} h_{x_i}(P) \wedge f_{x_i^P} \tag{4.23}$$

Example: With the example function f above, we define the Boolean function $h_{vol}^{[100,150]} = 1$ if $vol \in [100, 150]$, otherwise $h_{vol}^{[100,150]} = 0$. So the function f can be represented by the decomposition:

$$f(vol, p, t) = h_{vol}^{[100,150]} \wedge f_{vol^{[100,150]}}$$

We can represent the Boolean function $h(vol)$ by a simple decision diagram with one node having variable 'vol', an out-going edge with the predicate $[100, 150]$ as illustrated in the Figure 4.1.

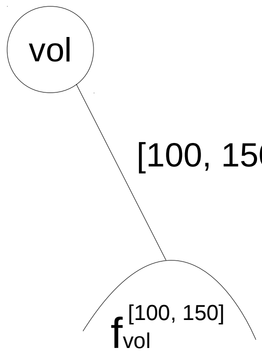


Figure 4.1: An example of the function decomposition

Each partial function $f_{x_i^P}$ can also be decomposed in respect to other variables, until it is independent from $\forall x_i \in X$. We can symbolize f as a decision diagram $G(V, E)$ with following properties:

- G is a rooted, directed acyclic graph (DAG) with the node set V having two types of nodes: internal nodes containing variables and leaf nodes containing boolean values.
- The internal node $v_{x_i} \in V$ has a variable $x_i \in D_i$ of the function f. Each out-going edge $e_{x_i} \in E$ represents the function in Eq. (4.18) over a partition $P_{x_i} \in \mathcal{P}(D_i)$. It states the clause: x_i *has the value in the range of the partition* P_{x_i}.
- Each sub-graph of the node v_{x_i} is a partial function $f_{x_i^P}$ in Eq. (4.23).

The Figure. 4.2 illustrates an example of G(V, E).

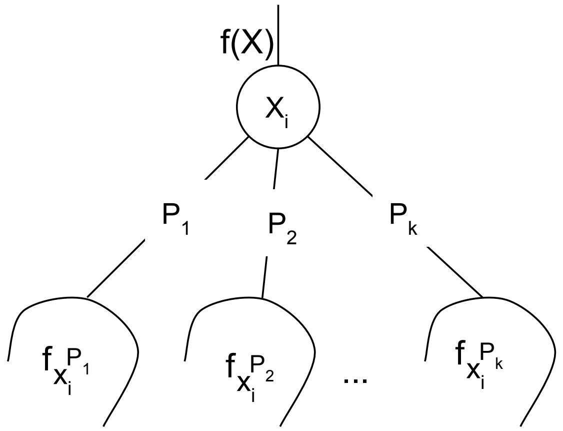


Figure 4.2: A decision diagram sample for the function decomposition

4.4.3 Multi-data-type Interval Decision Diagrams

The DAG G(V, E) can only represent a boolean function in Eq. (4.17). In Section 4.3, Match, AllOf, AnyOf and Target elements have the signature in Eq. (4.24), while Rule, Policy and Policy set elements have the signature in Eq. (4.25).

$$f : D_1 \times D_2 \ldots \times D_n \rightarrow V_M \tag{4.24}$$

$$f : D_1 \times D_2 \ldots \times D_n \rightarrow V_R \tag{4.25}$$

We extend G(V, E) to MIDD and X-MIDD representing Eq. (4.24) and Eq. (4.25) respectively.

Definition 4.3 (MIDD). MIDD is the G(V, E) representing a function having signature (4.24) over the V_M domain:

- Each internal node m in the MIDD is the tuple of $(x, s, \mathcal{C})$ in which x is the node variable, s is the state value: $s \in \{F, IN\}$. If x is marked as *critical*, $s = IN$, otherwise $s = F$.

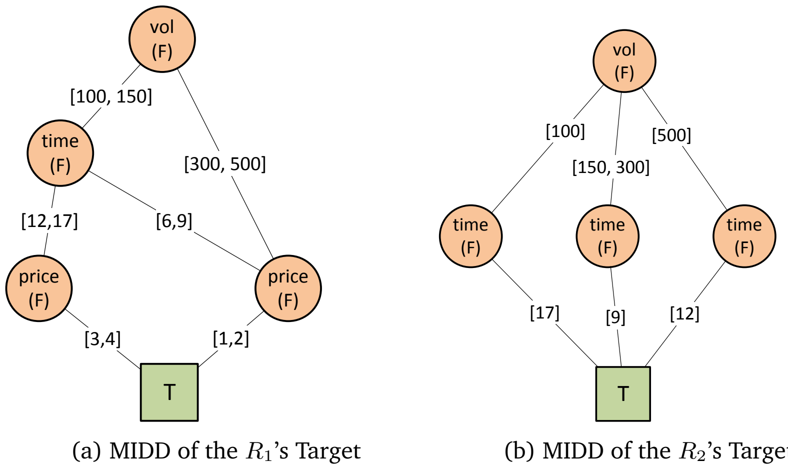
- The $\mathcal{C}$ is the set of tuples $(p, c) \in \mathcal{C}$, each represents an out-going edge containing a reduced interval partition p connecting m to a descendant node c.

- The descendant node c could either be another internal node or the external node containing T value. It is called the *T-leaf-node*.

- The evaluation of a request X against a MIDD is the traversal from the root node: at an internal node $(x_i, s, \mathcal{C})$, an out-going edge $(p, c) \in \mathcal{C}$ is selected if the value x_i of the request X belongs to the interval partition p: $x_i \in p$.

- In the evaluation process, if $\nexists x_i \in X$, the returned value is the state of the current internal node, which is either F or IN. The evaluation returns T if it reaches the *T-leaf-node*.

Examples of MIDDs can be found in the Figure 4.3.



(a) MIDD of the R_1's Target (b) MIDD of the R_2's Target

Figure 4.3: Sample MIDDs of the Target elements

Definition 4.4 (X-MIDD)**.** X-MIDD is the G(V, E) representing a function having signature (4.25) over the V_R domain:

- An internal node m is the tuple of $(x, s, \mathcal{O}, \mathcal{C})$: the state $s \in V_R$; $\mathcal{O}$ contains list of obligations and advices matching with s if $s \in \{P, D\}$, otherwise it is empty.

- An external node contains a policy evaluation result, which can be represented as a tuple of $(s, \mathcal{O})$ with s and $\mathcal{O}$ are similar to the internal node.

- The evaluation of the X-MIDD can be defined recursively in the Algorithm 4.1.

Examples of X-MIDDs can be found in the Figure 5.1.

We see that functions over V_R and V_M domains in section 4.3 have their set of operators. To facilitate the representation of such functions with MIDD and X-MIDD, we build equivalent algorithms in the next section.

4.5 Interval Processing and Decision Diagram Operations

This section defines interval processing and MIDD composition operations, which are used to create MIDDs from XACML elements.

```
Input: Request $X$ and the X-MIDD with the root $m$
Output: Evaluation decision
begin
    if ($m.x \notin X$) then
        return ($m.s, m.\mathcal{O}$);
    else
        foreach (($p, c) \in m.\mathcal{C}$) do
            if ($X[m.x] \in p$) then
                return x_eval($X, c$);
            end
        end
        return ($m.s, m.\mathcal{O}$);
    end
end
```

Algorithm 4.1: X-MIDD evaluation: x_eval function

4.5.1 Interval Partition Operations

Definition 4.5 (Reduced interval partition)**.** A reduced interval partition has least number of intervals compared to others having the same data ranges.

The reduction process of an interval partition is as follows: we find and combine intervals having adjacent ranges repeatedly until no adjacent-range interval is found. The result is the reduced interval partition.

While approach in [115] can only be used for integer data-type, our following definitions are more general and can support continuous data-types. Given two reduced interval partitions P_1 and P_2, we define operations as follows:

Definition 4.6 (Union)**.** $P = P_1 \curlyvee P_2$ has below properties:

- P is a reduced interval partition.
- All values belong to either partitions P_1 or P_2 also belong to P: $\forall v \in P_1 \cup P_2, v \in P$

Definition 4.7 (Intersect)**.** $P = P_1 \curlywedge P_2$ has the following properties:

- P is a reduced interval partition.
- P is composed from all common values of P_1 and P_2: $\forall v \in P_1 \cap P_2, v \in P$

Definition 4.8 (Complement)**.** $P = P_1 \ominus P_2$ is an interval partition that:

- P is a reduced interval partition.
- It contains values of P_1 but not P_2: $\forall v \in P_1 \setminus P_2, v \in P$

For example, with $P_1 = \{[-3, 4.5], [6.3, 8]\}$, $P_2 = \{(2, 5.1], (7.5, 9]\}$, we have:

- $P_1 \curlyvee P_2 = \{[-3, 5.1], [6.3, 9]\}$
- $P_1 \curlywedge P_2 = \{(2, 4.5], (7.5, 8]\}$
- $P_1 \ominus P_2 = \{[-3, 2], [6.3, 7.5]\}$

4.5.2 MIDD Operations

Given two functions f_1 and f_2 following the signature (4.24), we define conjunctive and disjunctive join algorithms representing operators in (4.2) and (4.3), respectively.

Variable ordering can affect the complexity of the MIDD, that we leave it for future work. Currently we choose an order in which variables appear in the policies. In the MIDD, the variable orders are lowest at the root and higher at deeper levels.

Let's call m_1 and m_2 are MIDDs for functions f_1 and f_2, the combining operators are shown in Algorithm 4.2 and 4.3, respectively.

We denote $m.P$ as the union of all partitions of node m's out-going edges: $m.P := \curlyvee\{\forall p, (p, c) \in m.\mathcal{C}\}$

4.5.2.1 MIDD Conjunctive Join

The algorithm representing conjunctive join operation in Algo. 4.2 is as follows:

- If either m_1 or m_2 is a *T-leaf-node*, the result is the other.
- Otherwise, if roots of m_1 and m_2 have the same variable, the root m of the result MIDD contains the same variable. The node state of m is the result of joining m_1 and m_2 states by the $\wedge$ operator. Child nodes of m are created by conjunctively joining children of m_1 and m_2 with equivalent intervals.
- If two inputs do not have the same variable, the children of the lower order (say l) are conjunctively joined with the higher order (say h) to create the descendants of the result MIDD.

4.5.2.2 MIDD Disjunctive Join

The Algorithm 4.3 for disjunctive join operator is quite similar. We add a new edge as the complement of all union-ed l's partitions to connect to h, meaning that if the value of $l.x$ in the request does not satisfy with $l.P$ predicate, then the decision is h. A special value $\perp$ is defined to handle the situation if the variable $l.x$ does not exist in the request X, the evaluation can traverse via this edge (lines 21-22).

In the Figure 4.4, we have three MIDDs constructed from parts of the R_1's target expression in Listing 4.1. Applying the Algorithm 4.3, we have the combined MIDD representing the target expression of the rule R_1 in Figure 4.3a. We note that the critical attribute setting of the variable *'vol'* in the first MIDD is transformed into the *'IN'* state. However, in the disjunctive combination the result has the 'F'' state due to the operation in the line 5 following Eq. (4.3): $F \vee IN \rightarrow F$

4.5.3 MIDD to X-MIDD Transformation

With conjunctive and disjunctive join algorithms, we can compose MIDDs representing logical expressions in *Target* and *Condition* elements. In the rule evaluation, we need to transform a MIDD over V_M domain into a X-MIDD over V_R based on (4.7) as follows:

```
Input: Two MIDDs : $m_1$ and $m_2$
Output: Conjunctive join $m = m_1 \wedge m_2$
begin
    if (any $m_i$ is a T-leaf-node) then
        return other $m_j$;
    else if ($m_1.x \equiv m_2.x$) then
        $s \leftarrow m_1.s \wedge m_2.s$;
        $m \leftarrow (m_1.x, s, \mathcal{C} := \emptyset)$;
        $P \leftarrow m_1.P \curlywedge m_2.P$;
        foreach (interval $I \in P$) do
            $c \leftarrow m_1.\mathcal{C}[I] \wedge m_2.\mathcal{C}[I]$);
            $p \leftarrow \{I\}$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup (p, c)$;
        end
    else
        $l \leftarrow m_i$ that has lower variable order;
        $h \leftarrow m_j$ that has higher variable order;
        $m \leftarrow (l.x, l.s, \mathcal{C} := \emptyset)$;
        foreach ($(p, c) \in l.\mathcal{C}$) do
            $c' \leftarrow c \wedge h$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{(p, c')\}$;
        end
    end
    return $m$;
end
```

Algorithm 4.2: The MIDD conjunctive join algorithm

- Join *Target* and *Condition* MIDDs using $\wedge$ operator.
- Replace the *T-leaf-node* by the *decision-leaf-node* containing $(e, \mathcal{O})$: e is the rule's effect, $\mathcal{O}$ contains applicable obligations and advices.
- For each internal node $m := (x, s, \mathcal{C})$, we map the state $m.s$ from $\{F, IN\}$ to V_R as follows:
  - If $m.s = F$, the new state is N.
  - If $m.s = IN$, the new state is IN_e with e is the rule's effect.

In [47], the critical attribute handling was implemented by the similar transforming function from MIDD to X-MIDD. However it required that critical settings of the same attribute in match expressions must be identical, which are either critical or non-critical. In this paper, we improve by the definition 4.3 in which the state value s can store critical settings of match expressions. This state value is then handled by all MIDD algorithms in section 4.5.2 and the transformation process from MIDD to X-MIDD.

4.5.4 X-MIDD Operations

Policy evaluation logic and XACML combining algorithms in sections 4.3.3.2 and 4.3.4 require operations over X-MIDDs, which are represented in following algorithms:

```
Input: Two MIDDs : $m_1$, $m_2$
Output: Disjunctive join $m = m_1 \vee m_2$
begin
    if (any $m_i$ is a T-leaf-node) then
        return T-leaf-node;
    else if ($m_1.x \equiv m_2.x$) then
        $s \leftarrow m_1.s \vee m_2.s$;
        $m \leftarrow (m_1.x, s, \mathcal{C} := \emptyset)$;
        $P \leftarrow m_1.P \curlyvee m_2.P$;
        foreach (interval $I \in P$) do
            $c \leftarrow m_1.\mathcal{C}[I] \vee m_2.\mathcal{C}[I]$;
            $p \leftarrow \{I\}$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{(p, c)\}$;
        end
    else
        $l \leftarrow m_i$ that has lower variable order;
        $h \leftarrow m_j$ that has higher variable order;
        $m \leftarrow (l.x, l.s, \mathcal{C} := \emptyset)$;
        foreach $((p, c) \in l.\mathcal{C})$ do
            $c' \leftarrow c \vee h$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{(p, c')\}$;
        end
        $p' \leftarrow (\{(-\infty, +\infty)\} \ominus l.P) \curlyvee \{\perp\}$ ;
        $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{(p', h)\}$;
    end
    return $m$;
end
```

Algorithm 4.3: The MIDD disjunctive join algorithm

4.5.4.1 Join *Target* with a Combined Children X-MIDD

The Algorithm 4.4 for policy evaluation Eq. (4.9) is almost similar to the MIDD conjunctive operator (Algorithm 4.2), with the differences that the join $\wedge$ is replaced by the $\overline{\wedge}$ operator.

4.5.4.2 Join X-MIDDs using XACML Combining Operators

Combining operators in Section 4.3.4 are used to join X-MIDDs, which are implemented in the Algorithm 4.5.

In the algorithm, if both m_1 and m_2 are *decision-leaf-nodes*, their decisions are combined using ω_{ca} operator with matching obligations and advices.

Otherwise, if roots of m_1 and m_2 have the same variable, the root of new X-MIDDalso contains this variable. The default returned decisions from m_1 and m_2 are combined using ω_{ca}. Its children is the combination of each of m_1 and m_2 children, respectively, aligned with each interval in the union interval partition $P = m_1.P \curlyvee m_2.P$.

If a m_i (say l) has lower variable order than the other (say h), we combine h with each child of l and add the output as the descendant of result MIDD m. We also add a new edge as the complement of all union-ed l's partitions to connect to h, meaning that if the value of $l.x$ in the request does not satisfy with $l.P$ predicate,

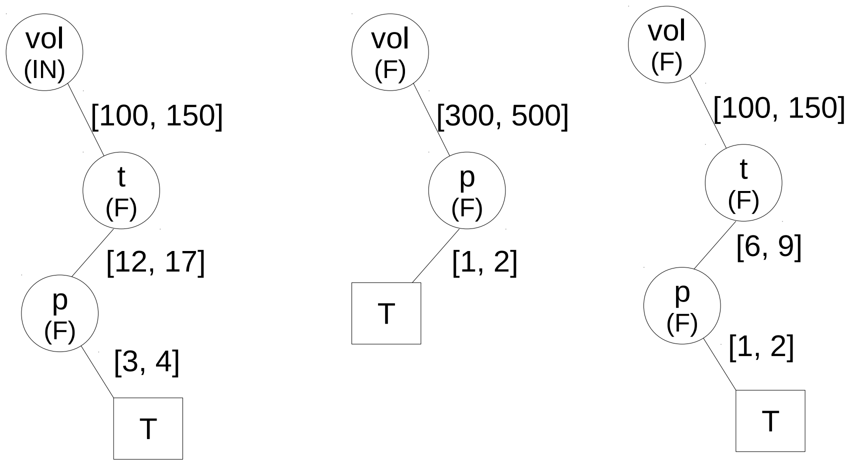


Figure 4.4: MIDDs of the R_0 target expression

```
   Input: A MIDD $m$ and a X-MIDD $\overline{m}$
   Output: A joined X-MIDD: $\overline{m}' = m \,\overline{\wedge}\, \overline{m}$
 1 begin
 2     if ($m$ is a T-leaf-node) then
 3         return $\overline{m}$;
 4     end
 5     if ($m.x \equiv \overline{m}.x$) then
 6         $s \leftarrow m.s \,\overline{\wedge}\, \overline{m}.s$;
 7         $\mathcal{O} \leftarrow (s \in \{P, D\})\ ?\ \overline{m}.\mathcal{O}(s) : \emptyset$;
 8         $\overline{m}' \leftarrow (m.x, s, \mathcal{O}, \mathcal{C} := \emptyset)$;
 9         $P \leftarrow m.P \curlywedge \overline{m}.P$;
10         foreach (interval $I \in P$) do
11             $c \leftarrow m.\mathcal{C}[I] \,\overline{\wedge}\, \overline{m}.\mathcal{C}[I])$;
12             $p \leftarrow \{I\}$;
13             $\overline{m}'.\mathcal{C} \leftarrow \overline{m}'.\mathcal{C} \cup \{(p, c)\}$;
14         end
15     else
16         $l \leftarrow m_i$ that has lower variable order;
17         $h \leftarrow m_j$ that has higher variable order;
18         $\overline{m}' \leftarrow (l.x, l.s, l.\mathcal{O}, \mathcal{C} := \emptyset)$;
19         foreach $((p, c) \in l.\mathcal{C})$ do
20             $c' \leftarrow c \,\overline{\wedge}\, h$;
21             $\overline{m}'.\mathcal{C} \leftarrow \overline{m}'.\mathcal{C} \cup \{(p, c')\}$;
22         end
23     end
24     return $\overline{m}'$;
25 end
```

Algorithm 4.4: Join algorithm following policy evaluation (4.9)

then the decision is h. A special value $\perp$ is defined to handle the situation if the variable $l.x$ does not exist in the request X, the evaluation can traverse via this edge.

```
Input: X-MIDDs $m_1, m_2$; combining algorithm $ca$
Output: Combined X-MIDD: $m = \omega_{ca}(m_1, m_2)$
begin
    if (all $m_i$ are decision-leaf nodes) then
        $s \leftarrow \omega_{ca}(m_1.s, m_2.s)$;
        $\mathcal{O} \leftarrow m_1.\mathcal{O}(s) \cup m_2.\mathcal{O}(s)$;
        return $(s, \mathcal{O})$;
    end
    if ($m_1.x \equiv m_2.x$) then
        $s \leftarrow \omega_{ca}(m_1.s, m_2.s)$;
        $\mathcal{O} \leftarrow m_1.\mathcal{O}(s) \cup m_2.\mathcal{O}(s)$;
        $m \leftarrow (m_1.x, s, \mathcal{O}, \mathcal{C} := \emptyset)$;
        $P \leftarrow m_1.P \curlyvee m_2.P$;
        foreach (interval $I \in P$) do
            $c \leftarrow \omega_{ca}(m_1.\mathcal{C}[I], m_2.\mathcal{C}[I])$;
            $p \leftarrow \{I\}$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{(p, c)\}$;
        end
    else
        $l \leftarrow m_i$ that has lower variable order;
        $h \leftarrow m_j$ that has higher variable order;
        $m \leftarrow (l.x, l.s, l.\mathcal{O}, \mathcal{C} := \emptyset)$;
        foreach ($(p, c) \in l.\mathcal{C}$) do
            $c' \leftarrow \omega_{ca}(c, h)$;
            $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{p, c'\}$;
        end
        $p' \leftarrow (\{(-\infty, +\infty)\} \ominus l.P) \cup \{\perp\}$;
        $m.\mathcal{C} \leftarrow m.\mathcal{C} \cup \{p', h\}$;
    end
    return $m$;
end
```

Algorithm 4.5: The algorithm for combining operators

4.6 Applications

Our main objective is to propose a high performance policy evaluation mechanism that can be applied in our access control approach for cloud. We illustrate that it can be solved by applying MIDD techniques in the next chapter. Beside that, we also point out that our mechanism can be reused in other policy management problems. At first, we define the following concept:

Given a policy P, let's define $|P|_e$ is the set of requests that policy evaluation is e with $e \in V_R$: $\forall X \in |P|_e, P(X) = e$.

Definition 4.9 (Policy subset)**.** Given two policy-trees P_1 and P_2 using the same attribute profile $\{a_1, \dots a_n\}$, P_1 is called the subset of P_2, denoting as $P_1 \subset P_2$ when $\forall e \in V_R \setminus N, |P_1|_e = |P_2|_e$.

We can see that $|P|_e$ essentially is the set of traversed paths from the root of the equivalent X-MIDD to the decision having value e.

Using the definition 4.9, we can point out following applications of the MIDD mechanisms:

- Policy testing: An important task in authorization policy testing is to enumerate all permit/deny or any decision $e \in V_R$ requests that a given policy can yield. It can be done by transforming this policy in to X-MIDD, then enumerating all traverse paths from the X-MIDD's root to a node containing decision value.

- Policy comparison: We can compare if the policy $P_1 \subset P_2$ by transforming them into X-MIDDs m_1 and m_2, enumerating all possible paths from m_1's root to decisions then make sure these paths also exists in P_2. The complexity of this problem is the complexities of transforming two policies (which are analyzed later) and the tree traversal problem.

- Reverse queries: the X-MIDD allows us to answer authorization queries in reverse orders: given a partial request, return the missing attributes and possible values that the policy can yield permit or deny decisions. A familiar sample reverse query could be "which resources can the subject Alice access during 9am-6pm?". Using X-MIDD the problem becomes enumerating all possible paths reaching the *permit* decision for the partial request {'Alice', 'read', time $\in$ [9, 18]}.

4.7 Conclusions

In this chapter, we analyze the logic behind XACML components and their evaluations. While XACML language is convenient to compose and manage authorization policies, there's no efficient implementation mechanism used for policy evaluation and analysis. For such purpose, we define a new data structure based on the decision diagram concept, known as the MIDD, along with operations on interval processing and combining MIDD algorithms. These mechanisms can be applied to substantially improve evaluation performance of the PDP engine, which will be shown in the next chapter.

Chapter 5

High Performance XACML Policy Evaluation

This chapter is based on the following publications:

- C. Ngo, M. X. Makkes, Y. Demchenko, and C. de Laat, “Multi-data-types interval decision diagrams for XACML evaluation engine," in Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on, 2013, pp. 257–266 [47].

- C. Ngo, Y. Demchenko, and C. de Laat, “Decision Diagrams for XACML Policy Evaluation and Management," In Computers & Security 49 (2015), pp. 1–16 [83].

5.1 Introduction

Access control systems for clouds should require high performance request throughput. Motivated from requirements of our proposals using XACML [48, 49, 97] and state-of-the-art of policy XACML engines and approaches [96, 98, 99], in this chapter we design and implement a novel high performance XACML engine. Compare to prior work, it is the most complete work on policy evaluation with important XACML 3.0 features that are event absent from others before.

Formulated from the XACML logical analysis in Chapter 4, our engine distinguishes from prior work by the following contributions:

- Support complex comparison functions for continuous data-types: the MIDD mechanism allows us to transform policies with inequality comparisons as in Listing 4.1.

- Handle all logical expression forms in the policies.

- Preserve original evaluation semantics in XACML elements and combining algorithms, not only in no-error scenarios but also in cases with “*not-applicable*" and “*indeterminate*" values in XACML 3.0, which is also compatible for version 2.0.

- Support *Condition* elements processing.
- Allow to define critical and non-critical attributes by the “*MustBePresent*" flag which is compliant with the XACML standard.
- Handle obligation and advice expressions in XACML elements.

Based on the refined analysis in section 4.3, our new algorithms are designed to follow evaluation standard: given a policy to be evaluated, first we create a MIDD for the target element; X-MIDDs of children rules are combined using the Algorithm 4.5; after that we join the target's MIDD with the combined X-MIDD using the Algorithm 4.4.

At first, we present the transformation process from regular XACML policies into decision diagram X-MIDDs in Section 5.2, and after that is the evaluation process in Section 5.3. We analyze the complexities of our approach in Section 5.4, then an implementation and experiments in Section 5.5. Finally, we conclude the chapter in section 5.6.

5.2 XACML Policy Transformations

In this section, by utilizing above defined operations, we solve the XACML evaluation problem by parsing and transforming XACML policies or policysets into X-MIDDs having equivalent evaluation semantics.

Given a XACML policy tree, our approach to construct an equivalent X-MIDD is as follows:

- Extract intervals and build MIDDs representing *Target* and *Condition* elements.
- Create the X-MIDD for each rule from its MIDDs following the mechanism in section 4.5.3.
- For a policy or policyset, join X-MIDDs of its children by equivalent combining operators to construct the final X-MIDD instance representing the root policy.

The final X-MIDD is then used for policy evaluation against incoming requests, which can be illustrated in Figure 5.2.

5.2.1 Creating MIDD from the Target Element

As described in Section 4.3, a match expression m is a tuple of (f, v, x) with the variable x containing a state $s \in F, IN$ representing the "*MustBePresent*" setting. It can be represented by a MIDD with two nodes: the internal node $(x, s, \mathcal{C} := \{I, c\})$, in which the interval $I = \{i | f(v, i) \to true\}$; c is the external *T-leaf-node*.

In the algorithm 5.1, the MIDD of an *AllOf* element having a list of *Match* can be composed quickly by aggregating intervals of *Match* MIDDs having the same variable x_i using function $I' = restrict(I, f, v) := \{i | i \in I, f(v, i) \to true\}$. For example, if $I = (-5, 8)$, then $restrict(I, \geq, 3) \to [3, 8)$.

From list of intervals il having exactly an interval per variable, we build a MIDD that has only a path from the root to a *T-leaf-node*. Each node n in the path has a variable x, joined state $sl[x]$ and an out-going edge having an interval $il[x]$.

```
Input: List of Match elements: $m = \{m_1, m_2.., m_k\}$
Output: Result MIDD: $m_1 \wedge m_2 \cdots \wedge m_k$
begin
    $il \leftarrow \emptyset; sl \leftarrow \emptyset$;
    foreach ($m_i \in m$) do
        $I \leftarrow (il[m_i.x] = null) ? (-\infty, +\infty) : il[m_i.x]$;
        $s \leftarrow (sl[m_i.x] = null) ? F : sl[m_i.x]$;
        $il[m_i.x] \leftarrow restrict(I, m_i.f, m_i.v)$;
        $sl[m_i.x] \leftarrow s \wedge m_i.x.s$;
    end
    $root \leftarrow null; tail \leftarrow null$;
    foreach (attribute $x \in sort(il.keys)$) do
        $n \leftarrow (x, sl[x], \mathcal{C} := (il[x], null))$;
        if ($root = null$) then
            $root \leftarrow n; tail \leftarrow root$;
        else
            $tail.addChild(n); tail \leftarrow n$;
        end
    end
    $tail.addChild(T\text{-}leaf\text{-}node)$;
    return $root$;
end
```

Algorithm 5.1: The *parseAllOf* function

The algorithm 5.2 creates the MIDD for a *Target* element using conjunctive and disjunctive join MIDD operators.

```
Input: List of AnyOf expressions: $A = \{a_1, a_2, ..a_k\}$
Output: A MIDD $t = a_1 \wedge a_2 \cdots \wedge a_k$
begin
    $t \leftarrow T\text{-}leaf\text{-}node$;
    foreach (AnyOf element $a \in A$) do
        $d \leftarrow T\text{-}leaf\text{-}node$;
        foreach (AllOf element $b \in a$) do
            $d' \leftarrow parseAllOf(b)$;
            $d \leftarrow d \vee d'$;
        end
        $t \leftarrow t \wedge d$;
    end
    return $t$;
end
```

Algorithm 5.2: The *parseTarget* function

Using this algorithm, the MIDDs of Target elements in rules R_1 and R_2 in the Listing 4.1 are shown in Figure 4.3.

5.2.2 Creating MIDD from a Condition Element

Given a request X and a *Condition* element $\kappa(X)$ in Eq. (4.5), we create a temporary variable $c := \kappa(X)$. In this case, the *Condition* element is similar to a *Match* element with the signature $(=, T, c)$ and can be represented by the MIDD having an internal node $(c, s, \mathcal{C} := \{([T], e)\})$ connecting to the *T-leaf-node* e. The state s receives IN value whenever there's a critical attribute in the expression, otherwise it is F value. We denote $parseCondition(cond)$ as the function to create the MIDD from a *Condition* element. It will be used in the process creating X-MIDD representing the rule evaluation.

5.2.3 Creating X-MIDD for a XACML Policy Tree

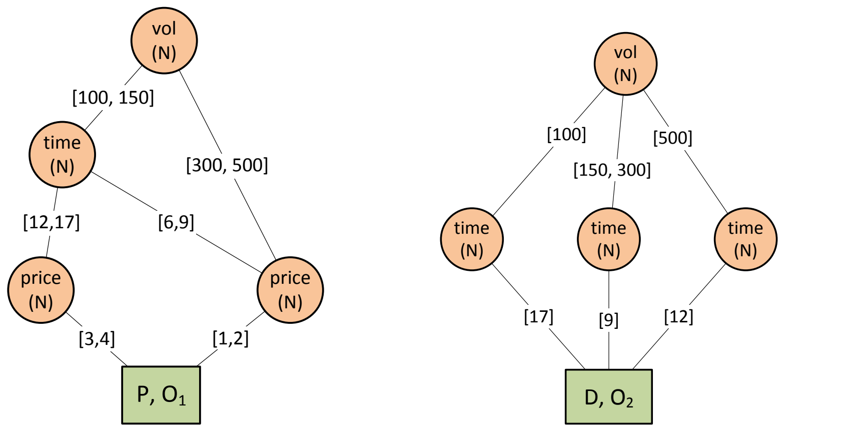
XACML policies are organized as a tree in which a node can be a rule, policy or policyset. Given a XACML policy node, the algorithm 5.3 is used to create its equivalent X-MIDD as follows:

- If this is a rule, we create MIDDs for *Target* and *Condition*, then transform the result into the rule's X-MIDD following the Section 4.5.3 using $transform$ function.
- If the node is either a policy or a policyset, based on the policy evaluation in Section 4.3.3.2, we parse its children recursively to obtain X-MIDDs and combine them using ω_{ca} in algorithm 4.5. The combination is then joined to the node's *Target* MIDD using algorithm 4.4. The result is the X-MIDD representing the policy node.

```
Input: A policy node in the policy tree: $n$
Output: X-MIDD representing the subtree: $\overline{m}$
begin
    $d_t \leftarrow parseTarget(n.t)$;
    if (n is a rule) then
        $d_c \leftarrow parseCondition(n.C)$;
        $d \leftarrow d_t \wedge d_c$;
        $\overline{m} \leftarrow transform(d, n.e, n.\mathcal{O})$;
    else
        $\psi \leftarrow null$;
        foreach ($n_i \in n.children$) do
            $d \leftarrow parsePolicyNode(n_i)$;
            $\psi \leftarrow \omega_{ca}(\psi, d, n.\mathcal{O})$;
        end
        $\overline{m} \leftarrow d_t \,\bar{\wedge}\, \psi$;
    end
    return $\overline{m}$;
end
```

Algorithm 5.3: The *parsePolicyNode* function

From MIDDs in Figure 4.3, using *transform* function we have equivalent X-MIDDs in Figure 5.1.



(a) X-MIDD of the R_1 (b) X-MIDD of the R_2

Figure 5.1: X-MIDDs of rules R_1 and R_2

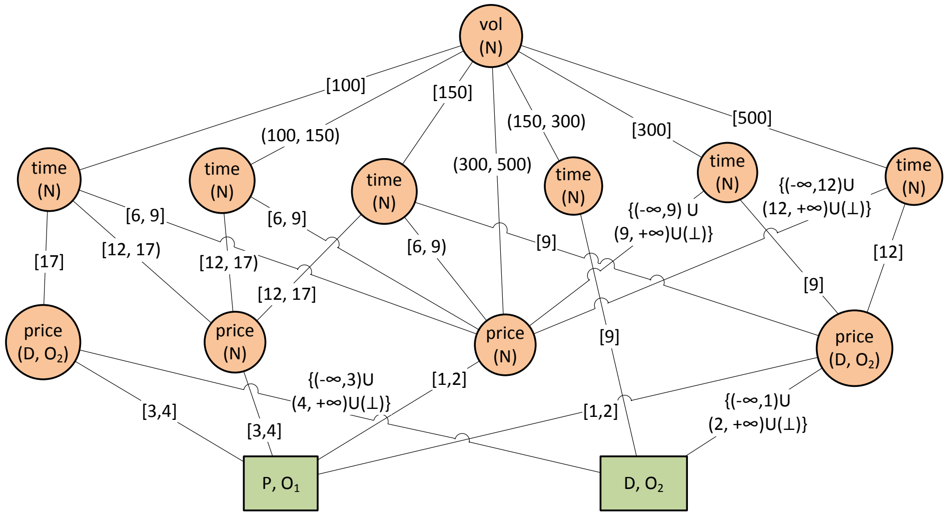


Figure 5.2: X-MIDD represents the sample policy P_0

Using the algorithm 5.3 starting from the root, we can create a X-MIDD representing the policy tree. For the sample in Listing 4.1, we have the X-MIDD in Figure 5.2.

5.3 XACML Policy Evaluations

5.3.1 Single-valued Request Evaluation

According to in XACML 3.0 standard [36], the evaluation process at the PDP is to find and combine decisions of applicable rules and policies in a policy tree for a given authorization request. Our transformation process in Section 5.3 allows to construct an X-MIDD structure having equivalent evaluation semantics with the policy tree. The evaluation process of a single-valued request $X = \{x_1, x_2 ..., x_n\}$ against the X-MIDD m is as follows:

- The traverse is from the root of m to a leaf-node, or when no attribute value is found.
- At an internal node m_i, if the value of the attribute $m_i.x$ exists in the X, find a matched out-going edge $(p, c) \in m_i.\mathcal{C}$ from m_i and continue the traverse from the child node c.
- If either no matched out-going edge found, or X does not contain value of $m_i.x$, the evaluation return the node's state $m_i.s$.
- If the current traverse node c is a leaf-node, returns its decision $(s, \mathcal{O})$.

A single-valued request X is defined as a list of attribute values, exactly one value per attribute, $X = \{x_1, x_2 ..., x_n\}$. Evaluating X against policies is to find the matching path in the equivalent X-MIDD. At a node, finding the matching edge can be done either using sequential or binary searches, since out-going edges of an internal nodes having ordered interval partitions. We currently support single-valued requests.

If the matching path found, the engine reaches the leaf node. Here, if the condition C is "*true*", the result is the decision value and equivalent objects (i.e., obligations, advices). Otherwise, it returns N. If the matching path is not found due to missing attributes or no applicable edge at the node n, the evaluation return $n.s \in V_R$, the default returned-decision.

5.3.2 Multi-valued Request Evaluation

XACML allows multi-valued attribute requests, where an attribute can store a list of values (e.g., a person can have several roles: employees, managers, etc). However, current XACML behavior on multi-valued requests processing has some concerns. With policy in Listing 4.1, given a request $X = \{150, \{10, 19\}, 4\}$ with the *time* attribute can either *10am* (10) or *7pm* (19), the evaluation following XACML standards [36, 38] such as SunXACML [82] claims that R_1 is the applicable rule. It has such result because each value in the bag of *time* attribute $\{10, 19\}$ is checked with each match expressions: the value 10 passes the condition $time < 17$, and value 19 passed the condition $time > 12$. While in practical, we expect R_1 is not applicable when it requires a *time* value passes both conditions, not two separate values.

[98] and [99] decomposed multi-valued requests in set of single-valued requests to evaluate and combine all of applicable rules. As illustrated in the above example, this processing differs from original standards.

For multi-valued policies supports in approaches of [98, 99], we argue that they only used in XACML condition expressions, not in target expressions due to the value v in $m_k := (x, f, v)$ can only receive a literal value [36, 38]. Because they only deal with target expressions, their supports are redundant. Our solution replaces condition expressions as variables, so it is possible to handle multi-valued attributes in policies.

5.4 Analysis

5.4.1 Features Comparison

Based on logical analysis in Section 4.3, our proposed mechanism covers most of missing XACML features from prior works (e.g in [98] and [99]):

- We have succeeded to fully support XACML logical expressions analyzed in Section 4.3 with multiple data-types and comparison operators.
- Preserving original combining algorithms semantic in handling indeterminate and not-applicable states: prior work could handle simple *Permit* or *Deny* decisions, but incorrectly for others.
- Critical attribute setting: to the best of our knowledge, we are the first work to support this feature with high performance evaluation.

5.4.2 Complexities

5.4.2.1 Space complexity

We suppose that a policy-tree uses n attributes $\{a_1, a_2.., a_n\}$, $a_i \in P_i$. Assuming that the domain P_i has k_i different values appearing in the policies, so P_i can be separated into at most $2k_i + 1$ intervals or partitions, including open and degenerate intervals. The X-MIDD representing the policy tree has at most $2k_i + 1$ out-going edges from any node at level l_i.

With $n + 1$ levels, the largest number of nodes at level l_i of the X-MIDD is $\prod_{j=1}^{i} (2k_j + 1)$. Therefore, the worst-case space complexity is $\mathcal{O}(\sum_{i=1}^{n} \prod_{j=1}^{i} (2k_j + 1))$.

It shows that the space complexity in the worst case does not depend on neither the policies size, height of policy tree, nor the complexity of logical formulas in their target expressions. It only depends on the number of attributes and number of distinct attribute values in the policy tree. Similar to the BDD approach, the size of the MIDD is affected heavily by the attribute ordering [114]. The implementation shows that our algorithms have efficient performance with reasonable graph size.

5.4.2.2 Evaluation time complexity

The policy evaluation process in our X-MIDD is the traversal from the root to a leaf node or an internal node where it cannot find any applicable out-going edges. At the level l_i of the X-MIDD, it has to find an applicable item among at most $2k_i + 1$ out-going edges. Using the sequential search, the evaluation time complexity is $\mathcal{O}(\sum_{i=1}^{n}(2k_i + 1))$. For binary search it is $\mathcal{O}(\sum_{i=1}^{n}\lfloor\log_2(2k_i + 1) + 1\rfloor)$.

Similar to the space complexity, it is shown that the evaluation time complexity only depends on number of attributes n and number of appearing attribute values $k_{i|i=1..n}$. It is the advantage of the proposed approach to evaluate a large number of policies containing complex logical expressions. However, the drawback is the memory cost with policies having a large number of n and k_i values.

Our approach has the time complexity does not depend on number of policies, but may not handle well on systems with high number of attributes. Actually, numbers of attributes in authorization systems are often quite limited, could be up to about 10 attributes. However, the number of policies usually expands in proportion to the organization and system scale. Therefore, the proposed approach is still useful in practical.

5.5 Implementation and Evaluation

In our experiments, we compare our implementation with the standard SunXACML engine [82].

We do not make direct experiments to compare with prior work [98, 99]. The work in [98] has better performance in datasets using only equality operators due to numerical comparisons are faster. In other cases, it does not support datasets with inequality operators. The approach in [99] does not publish neither its implementation nor datasets. However, we see that our X-MIDD structure has similar time complexity to their matching-tree (MT) in the worst case. But after MT evaluation, they need to evaluate the combining-tree (CT), that the time complexity relies on height of policy tree. So ours has somewhat better evaluation performance.

5.5.1 Environment and Datasets

We implement our SNE-XACML engine [85] in JRE 1.7. Experiments are carried out on a Linux x64 system with Intel i5 core 2.67 GHz and 4GB RAM. The datasets are XACML 3.0 policies.

Due to lacking of XACML 3.0 implementations, we can only compare our work with the popular XACML 2.0 engine, SunXACML. It means that sophisticate *indeterminate* decisions in v3.0 in our work cannot compare to decisions from SunXACML. This evaluation needs to be improved by conformance tests of XACML 3.0 in the future. Currently, such tests do not present in the XACML community.

In our experiments, we convert policies 3.0 back to version 2.0 in order to be compatible with the referenced SunXACML. All *indeterminate* values in version 3.0

are mapped to an *indeterminate* value in version 2.0.

We use three datasets in Table 5.1. The first one is a real-life policy taken from GEYSERS project [10] with some obligations and a critical attribute marked as *“MustBePresent=true*". The *continue-a* policy is taken and converted from [100]. And the *synthetic-360* is our randomly generated policy using 80% equality operator and 20% inequality operators. We also select random mixture of all combining algorithms.

We see that most target expressions in *continue-a* policy are trivial with either empty or a few match expressions in a level. The *synthetic-360* policy is more complex: each Target, AnyOf and AllOf elements contain from 0 to 4 children. It is the reason why the X-MIDD generated from the *synthetic-360* is more complex than from continue-a.

In our testbed, we generate requests randomly following uniform distribution for each datasets. They are then evaluated by two engines.

Table 5.1: Sample Policy Datasets

Datasets	Policy levels	#Policy sets	#Policies	#Rules	Attri-butes	Operators
GEYSERS [10]	3	6	7	33	3	=
Continue-a [100]	6	111	266	298	14	=
Synthetic-360 [47]	4	31	72	360	10	=(80%), co-mplex(20%)

5.5.2 Validation

Our empirical validation is as follows:

- Experiments are performed on the three given datasets in Table 5.1, being a public policy in related work [100], our synthetic policy [47] and a project policy [10].
- For each dataset, 1000 random requests are generated uniformly distributed over attribute value ranges.
- Generated requests are evaluated by two engines: the standard, popular XACML engine [82] and our XACML engine [85]. Outputs of two engines are expected to be the same.
- We run experiments multiple times.

Results of these empirical experiments confirm the correctness of our approach and implementation.

5.5.3 Performance Analysis

We run the testbed ten times, each evaluates a million requests by the SNE-XACML and a thousand ones by the SunXACML. The average response times of two engines

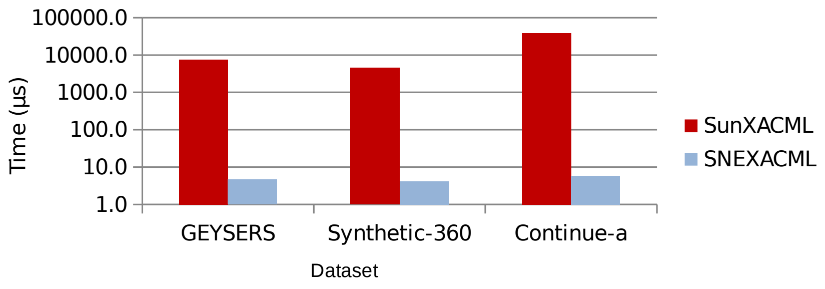


Figure 5.3: Average evaluation response times

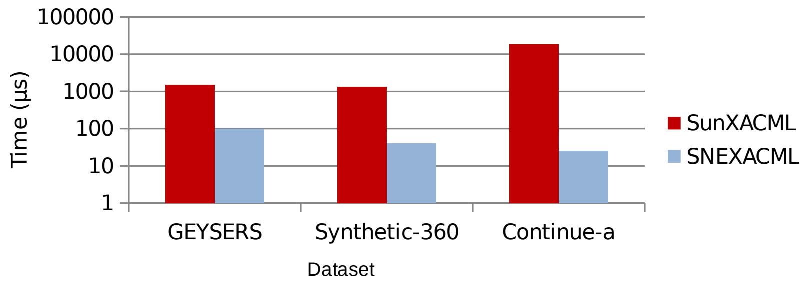
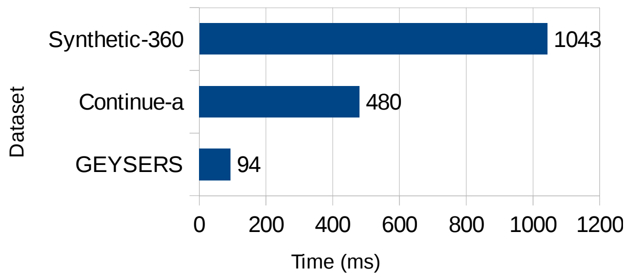


Figure 5.4: Standard deviation of evaluation response times

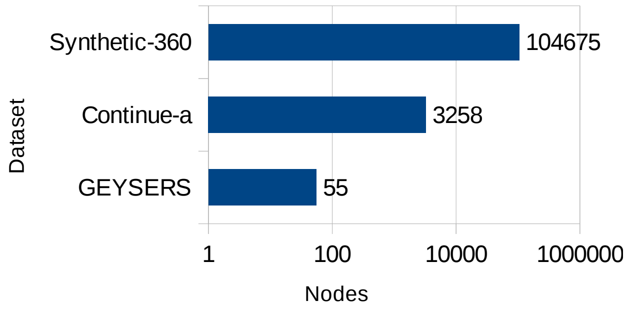
are shown in Figure.5.3. We observe that our engine is about three orders of magnitude faster than the SunXACML. Its response times are almost the same in three datasets, while the basic engine's performance is heavily dependent on the complexity of policies' logical expressions. Figure. 5.4 shows the standard deviation of evaluation response time. With all datasets, SunXACML has greater variation in response times than ours. This illustrates our analysis of evaluation time complexity, which is linear in attribute sizes and logarithmic complexity in attribute values.

Figure. 5.5 shows our micro-benchmark results. The pre-processing time and X-MIDD size are highly dependent on policy complexity, which is the number of attribute n and number of attribute values k. The X-MIDD of continue-a policy with 14 attributes has less nodes than the diagram of synthetic-360 policy with 10 attributes. Because the later is randomly generated, therefore it contains higher $\{k_i\}$ values.

The request evaluation time is composed from three parts: time to extract attribute values from XACML requests, time to evaluate these values on the X-MIDD representing the policy and time to create XACML response messages. These time fractions are shown in Figure. 5.6. We can see that the response conversion time fraction is negligible, while the request conversion time is quite remarkable compared to the X-MIDD evaluation time. In most cases, response messages only contain decisions, thus their sizes are usually small. However, sizes of request messages in the XML format depend on the number of attributes and attribute values. Therefore, request messages are much larger than response messages. The experiment shows that the request conversion times increase proportionally to the



(a) Pre-processing times



(b) Number of nodes in X-MIDD

Figure 5.5: SNE-XACML micro-benchmarks

number of attributes in three datasets.

Figure. 5.6 also illustrates that the X-MIDD evaluation fraction times are remarkable in the total response time when the number of attributes is small, e.g., in GEYSERS's dataset. For datasets with more attributes, the X-MIDD evaluation fraction times are in the range about 50%-60%. We can conclude that the conversion overhead is the bottleneck in ours due to the inefficient of XML parsing compared to the optimized policy evaluation. Note that the conversion overheads in SNE-XACML and other engines are quite similar because they often use popular XML processing mechanisms such as JAXB.

5.6 Conclusions

In this chapter, we applied our analysis results in Chapter 4 to design and develop a high performance XACML policy evaluation engine. It not only boosts the system performance, but also preserves original evaluation semantics, which is missed in prior work. The proposed solution could handle the complicated logical expressions defined in policies' predicates, correctness of combining algorithms semantics, critical attribute setting, obligations and advices handling. The evaluation and analysis show that the presented approach has the efficient performance in time complexity while having the reasonable space complexity. Experiments prove that our solution implemented in the open sourced SNE-XACML engine [85] has both

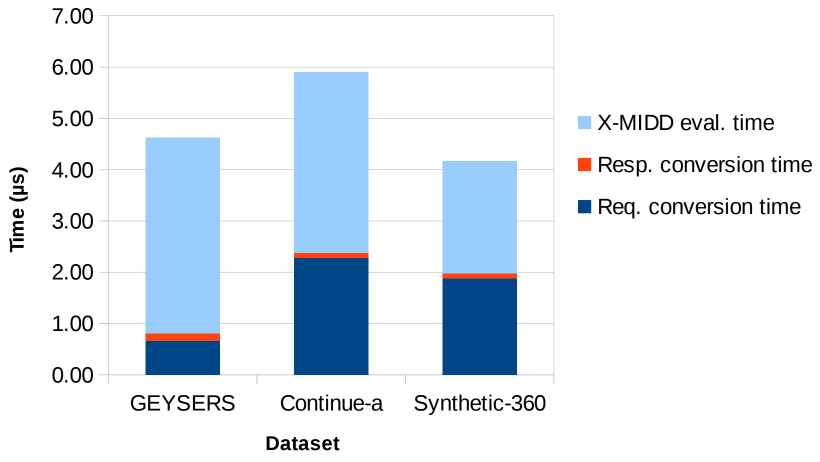


Figure 5.6: SNE-XACML evaluation time fractions

the significant performance compared to the referenced SunXACML engine [82] and validated evaluation results.

Chapter 6

Conclusion

In this chapter, we revisit questions in Chapter 1 and use research presented in Chapters 2, 3, 4 and 5 to answer them.

6.1 Answers to Research Questions

1. *How do we design a flexible and scalable access control model supporting the on-demand provisioned self-service of cloud infrastructure services?*

Chapter 2 answers this question. It introduced MT-ABAC, an ABAC model with multi-tenant properties integrating with the INDL cloud information model [79]. Inheriting the flexibility of ABAC, the proposed model can use attributes of subjects and resources in authorization policies. To support on-demand provisioned self-service, the model extended INDL by attaching policy templates binding with resource descriptions. Upon provisioning and rescaling, policies therefore can be generated and updated. Our prototype was implemented by an INDL extension for policy template descriptions. These templates were retrieved by SPARQL queries to compile into policies using a subset of the XACML language.

2. *How does the proposed access control model adapt the virtualized resource sharing for multiple customers, who can manage their own resources not only in a single domain but also in multiple distributed domains?*

Our proposed approach MT-ABAC in Chapter 2 extended the generic ABAC model to support multi-tenancy scenarios of the cloud systems. It distinguished general subjects into separated types, including providers, tenants and users. The capabilities and constraints of these subject types were defined to support the multi-tenancy features. This model therefore can manage virtualized resources pooling consumed by multiple customers at a cloud provider, where each of them could control authorization policies for their own subscribed resources.

To apply the MT-ABAC model for inter-provider scenarios, in Chapter 3 we provided token exchange approaches to solve distributed authorization and inter-domain security context management problems. In these cases, cloud resources can interconnect between multiple providers. Our approach contained grant-tokens to relay decisions from the tenant's domain to the provider's domain via end-users. Upon validating grant-tokens, the system issued access-tokens for each provider's domain, which then can be used by users to access resources. In this way, our approach allowed users to establish dynamic trust relationships with the chain of indirect providers for a specific authorization context.

3. *How do we implement a high performance authorization policy evaluation engine, which should be required in access control solutions for cloud providers?*

By applying XACML as the *de-facto* access control policy standard in our model, we provided the flexibility and expressiveness of the language in policy composition and management. However, to optimize XACML in high performance systems for cloud providers, our work in Chapter 4 analyzed the complex logic behind XACML components and evaluation processes. We then proposed the MIDD, a decision diagram-based mechanism with interval partition operations and MIDD operators. Our data structures and algorithms demonstrated in Chapter 5 to parse XACML policy tree into decision diagrams which substantially improve evaluation performance of the policy engine and the access control system. The proposed mechanism is reusable when it is able to apply not only for the policy evaluation, but also in various policy management problems.

6.2 Discussion

In this thesis, we identified challenges, models and mechanisms in building access control systems for cloud resource management at providers. Our solution specialized to use the ABAC model into multi-tenancy scenarios, with necessary constraints and capabilities for policy management. For practical implementations, we chose the *de-facto* attribute-based policy language XACML to apply in our model. Using XACML would make the approach having better compatibility and re-usability for different systems, not only in our project testbeds GEYSERS for cloud IaaS management, but also other cloud services systems.

The main drawback of XACML in applying for cloud system managements is performance issues for large-scale multi-tenant environments. Due to complex logic behind policies and algorithms, existing approaches cannot adapt high throughput requirements of cloud systems with thousands of customers. We solved this problem by analyzing XACML logic to propose data structures and algorithms for a high performance XACML policy evaluation engine. The implementation proved that our solution while still maintained policy semantics, can handle request throughputs with magnitudes faster than the standard implementation. Applied in the MT-ABAC for cloud infrastructure services of the DACI system, the testbed showed that the throughput can reach about 1400 requests/s for a scenario with 1000 customers, each having a separated, non-trivial virtual infrastructure. It is possible to not only scale-up the authorization systems with more robust computing powers, but also horizontally scale-out to increase overall system throughput because the policy evaluation mechanisms in our systems are stateless.

Our testbed was implemented and integrated successfully to the cloud infrastructure services in the GEYSERS project [10] for single and multiple providers scenarios. The design abstractions between cloud resource managements at the composition layers (a.k.a LICL layers in the GEYSERS project) and the virtual resource managements allow our approach to be able to integrate with different IaaS cloud stacks, not only the OpenNebula [84] in GEYSERS, but also others such as OpenStack or Eucalyptus. However, integrating with such stacks needs further real deployments and experiments. In other aspects, even though we tried to design MT-ABAC in the DACI system independent with cloud service models, our work contained primarily the testbed for IaaS model as in GEYSERS, while did not have validation work for PaaS and SaaS. Therefore, we have spaces to investigate and improve for other cloud services, which could be done in our future work.

6.3 Future Research

We propose two directions in the future research. The first line is to research on integrations and compatibilities of the MT-ABAC with other cloud service models and legacy systems. The second line is to analyze and compose access control policies using our techniques in Chapters 4 and 5.

Regarding integrating MT-ABAC to cloud systems providing other service models, resource description models managing cloud resources should be capable of on-demand provisioning and rescaling. Provider policies should be then generated and synchronized according to cloud provisioning and rescaling. Depending on desired scenarios between providers, tenants and end-users, related policies constraints should be defined and enforced at policy managements. In case of multi-domains scenarios, our token exchanges approach can be extended to establish, manage and validate trust and delegation management.

There are also open research on how to integrate and interoperate legacy access control systems at tenants' sides with the MT-ABAC. Authorization rules at legacy systems while can be composed by various policy languages should be validated and verified by defined constraints in MT-ABAC. The research should investigate on transforming legacy rules into the XACML, which facilitate for constraint verification and validation.

In other aspects, the MIDD data structures and algorithms used to solve the XACML evaluation performance problem can be applied to solve policy analysis and verification problems [100, 102]. Approaches in [100, 102] limited only to a subset of XACML language, while ours provide the most comprehensive policy semantic features. Other problems mentioned in section 4.6 such as policy testings, policy comparison, reverse authorization queries are also in our future researches on policy managements. Another section is to actively develop our open source XACML policy evaluation engine with enhanced features and profiles.

The development of cloud systems are becoming more popular. As a result, access control management for such systems should be aware to research further. Our work in this thesis regarding models, mechanisms and testbeds for access control are small but prospective contributions not only for Cloud Computing but also for general access control research.

Acronyms

AA	Attribute Authority.
ABAC	Attribute-based Access Control.
ABE	Attribute-based Encryption.
ADF	Access Control Decision Function.
AEF	Access Control Enforcement Function.
AWS	Amazon Web Services.
BDD	Binary Decision Diagram.
CIM	Common Information Model.
CT	Combining Tree.
DAC	Discretionary Access Control.
DACI	Dynamic Access Control Infrastructure.
DAG	directed acyclic graph.
FIA	Fine-grained Integration Algebra.
IaaS	Infrastructure as a Service.
IAM	Identity and Access Management.
IDD	Interval Decision Diagram.
INDL	Infrastructure and Network Description Language.
LDAP	Lightweight Directory Access Protocol.
LICL	Logical Infrastructure Composition Layer.
MAC	Mandatory Access Control.

MIDD	Multi-datatype Interval Decision Diagram.
MT	Matching Tree.
MT-ABAC	Multi-tenant Attribute-based Access Control.
MT-RBAC	Multi-tenant Role-based Access Control.
MTBDD	Multi-Terminal Binary Decision Diagram.
MTIDD	Multi-Terminal Interval Decision Diagram.
NIST	US. National Institute of Standards and Technology.
PaaS	Platform as a Service.
PDP	Policy Decision Point.
PEP	Policy Enforcement Point.
PERMIS	PrivilEge and Role Management Infrastructure Standards.
PIP	Physical Infrastructure Provider.
RBAC	Role-based Access Control.
RDF	Resource Description Framework.
SaaS	Software as a Service.
SAML	Security Assertion Markup Language.
SLA	Service Level Agreement.
SWRL	Semantic Web Rule Language.
VI	Virtual Infrastructure.
VIO	Virtual Infrastructure Operator.
VIP	Virtual Infrastructure Provider.
VM	Virtual Machine.
VR	Virtual Resource.
X-MIDD	Multi-datatype Interval Decision Diagram for XACML.
XACML	eXtensible Access Control Markup Language.
XSD	XML Schema Definition.

Bibliography

[1] Ian Foster et al. “Cloud computing and grid computing 360-degree compared”. In: *Grid Computing Environments Workshop, 2008. GCE’08*. Ieee. 2008, pp. 1–10.

[2] Armando Fox et al. “Above the clouds: A Berkeley view of cloud computing”. In: *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28 (2009), p. 13.

[3] Tharam Dillon, Chen Wu, and Elizabeth Chang. “Cloud computing: issues and challenges”. In: *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee. 2010, pp. 27–33.

[4] CN Höfer and G Karagiannis. “Cloud computing services: taxonomy and comparison”. In: *Journal of Internet Services and Applications* 2.2 (2011), pp. 81–94.

[5] Peter M. Mell and Timothy Grance. *SP 800-145. The NIST Definition of Cloud Computing*. Tech. rep. Gaithersburg, MD, United States, 2011.

[6] M. D Hogan et al. *Cloud Computing Standards Roadmap*. Tech. rep. SP 500-291. NIST, 2011. URL: `http://www.nist.gov/manuscript-publication-search.cfm?pub_id=909024`.

[7] Amazon. *AWS Web Servicess (AWS)*. `http://aws.amazon.com`. 2013.

[8] Luis M Vaquero et al. “A break in the clouds: towards a cloud definition”. In: *ACM SIGCOMM Computer Communication Review* 39.1 (2008), pp. 50–55.

[9] Fang Liu et al. *NIST Cloud Computing Reference Architecture*. Tech. rep. SP 500-292. NIST, 2011.

[10] *GEYSERS - Generalised Architecture for Dynamic Infrastructure Services*. Tech. rep. The GEYSERS Project (FP7-ICT-248657), 2010. URL: `http://www.geysers.eu/`.

[11] Amazon. *Amazon Elastic Compute Cloud*. `http://aws.amazon.com/ec2/`. 2006.

[12] Microsoft. *Microsoft Azure*. `http://azure.microsoft.com/en-us/services/`. 2010.

[13] Google. *Google Compute Engine*. `https://cloud.google.com/compute/`. 2012.

[14] Rackspace. *The Rackspace Managed Cloud*. http://www.rackspace.com/cloud. 2010.

[15] Google. *Google App Engine*. https://cloud.google.com/products/app-engine/. 2013.

[16] Salesforce. *Heroku*. http://heroku.com/. 2007.

[17] Google. *Google Apps for Work*. https://www.google.com/work/apps/business/. 2006.

[18] Apple. *iCloud*. https://www.icloud.com/. 2013.

[19] Microsoft. *Microsoft Office 365*. http://office.microsoft.com/. 2011.

[20] *D1.4 - Initial Set and Technical Specification of System Requirements*. Tech. rep. The GEYSERS Project (FP7-ICT-248657), 2010. URL: http://www.geysers.eu/.

[21] *D1.1 - Identification, Description and Evaluation of the Use Case Portfolio and Potential Business Models*. Tech. rep. The GEYSERS Project (FP7-ICT-248657), 2010. URL: http://www.geysers.eu/.

[22] *Security Frameworks for Open Systems: Access Control Framework*. 1996. URL: http://pubs.opengroup.org/onlinepubs/009609199/chap3.htm#tagcjh_04_02_02.

[23] J. Vollbrecht et al. *RFC 2904 - AAA Authorization Framework*. Tech. rep. RFC 2094. Aug. 2000. URL: http://tools.ietf.org/html/rfc2904.

[24] Butler W Lampson. “Protection”. In: *ACM SIGOPS Operating Systems Review* 8.1 (1974), pp. 18–24.

[25] Carl E. Landwehr. “Formal models for computer security”. In: *ACM Computing Surveys* 13 (1981), pp. 247–278.

[26] David FC Brewer and Michael J Nash. “The chinese wall security policy”. In: *Security and Privacy, 1989. Proceedings., 1989 IEEE Symposium on*. IEEE. 1989, pp. 206–214.

[27] D Elliott Bell and Leonard J LaPadula. *Secure computer systems: Mathematical foundations*. Tech. rep. DTIC Document, 1973.

[28] R.S. Sandhu et al. “Role-Based Access Control Models”. In: *IEEE computer* 29.2 (1996), pp. 38–47. DOI: 10.1109/2.485845.

[29] David F Ferraiolo et al. “Proposed NIST standard for role-based access control”. In: *ACM Transactions on Information and System Security (TISSEC)* 4.3 (2001), pp. 224–274.

[30] *American national standard for information technology – role based access control*. Tech. rep. ANSI INCITS 359-2004. ANSI, 2004.

[31] D Richard Kuhn, Edward J Coyne, and Timothy R Weil. “Adding attributes to role-based access control”. In: *IEEE Computer* 43.6 (2010), pp. 79–81.

[32] V.N.L. Franqueira and R.J. Wieringa. “Role-Based Access Control in Retrospect”. In: *Computer* 45.6 (June 2012), pp. 81–88. ISSN: 0018-9162. DOI: 10.1109/MC.2012.38.

[33] Jose M Alcaraz Calero et al. “Toward a multi-tenancy authorization system for cloud services”. In: *Security & Privacy, IEEE* 8.6 (2010), pp. 48–55.

[34] Jorge Bernal Bernabe et al. “Semantic-aware multi-tenancy authorization system for cloud architectures”. In: *Future Generation Computer Systems* (2012).

[35] Bo Tang, Qi Li, and Ravi Sandhu. “A multi-tenant RBAC model for collaborative cloud services”. In: *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*. IEEE. 2013, pp. 229–238.

[36] OASIS. *XACML v3.0: Core Specification*. Jan. 2013. URL: `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf`.

[37] Amazon. *AWS Identity and Access Management (IAM)*. `http://aws.amazon.com/iam/`. 2013.

[38] OASIS. *eXtensible Access Control Markup Language XACML Version 2.0*. 2005. URL: `http://docs.oasis-open.org/xacml/2.0/access\_control-xacml-2.0-core-spec-os.pdf`.

[39] Nicodemos Damianou et al. “The ponder policy specification language”. In: *Policies for Distributed Systems and Networks*. Springer, 2001, pp. 18–38.

[40] *Ponder: A Policy Language for Distributed Systems Management*. URL: `http://www-dse.doc.ic.ac.uk/Research/policies/ponder.shtml`.

[41] David W Chadwick and Alexander Otenko. “The PERMIS X. 509 role based privilege management infrastructure”. In: *Future Generation Computer Systems* 19.2 (2003), pp. 277–289.

[42] *X.509|ISO/IEC 9594-8, The Directory: Authentication Framework*. 2000.

[43] *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0, OASIS Standard*. SAML. OASIS, Mar. 2005. URL: `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`.

[44] Frederick Chong, Gianpaolo Carraro, and Roger Wolter. *Multi-Tenant Data Architecture*. `http://msdn2.microsoft.com/en-us/library/aa479086.aspx`. Microsoft, June 2006.

[45] Yuri Demchenko et al. “On-demand provisioning of cloud and grid based infrastructure services for collaborative projects and groups”. In: *Collaboration Technologies and Systems (CTS), 2011 International Conference on*. IEEE. 2011, pp. 134–142.

[46] Eduard Escalona et al. “GEYSERS: a novel architecture for virtualization and co-provisioning of dynamic optical networks and IT services”. In: *Future Network & Mobile Summit (FutureNetw), 2011*. IEEE. 2011, pp. 1–8.

[47] Canh Ngo et al. “Multi-data-types interval decision diagrams for XACML evaluation engine”. In: *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*. IEEE. 2013, pp. 257–266.

[48] Canh Ngo et al. “Security Framework for Virtualised Infrastructure Services Provisioned On-demand”. In: *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*. 2011, pp. 698–704. DOI: `10.1109/CloudCom.2011.108`.

[49] Canh Ngo et al. “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures”. In: *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*. IEEE. 2012, pp. 343–349.

[50] Canh Ngo, Yuri Demchenko, and Cees de Laat. “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services”. In: *Journal of Information Security and Applications* (Accepted 2015).

[51] OGF ISOD-RG. *OGF Infrastructure Service On-demand Provisioning Research Group*. URL: `http://www.ogf.org/gf/group_info/view.php?group=ISOD-RG`.

[52] OASIS IDCloud TC. *OASIS Identity in the Cloud*. URL: `http://wiki.oasis-open.org/id-cloud/`.

[53] Shankar Babu Chebrolu, Vinay Bansal, and Pankaj Telang. “Top 10 cloud risks that will keep you awake at night”. In: *CSICO, available at: `https://www.owasp.org/images/4/47/Cloud-Top10-Security-Risks.pdf`* ().

[54] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. “Security and Privacy Challenges in Cloud Computing Environments.” In: *IEEE Security & Privacy* 8.6 (2010), pp. 24–31.

[55] D. Catteddu and G. Hogben. *Cloud Computing Risk Assessment*. Tech. rep. ENISA, 2009. URL: `http://www.enisa.europa.eu/activities/risk-management/files/deliverables/cloud-computing-risk-assessment`.

[56] *GEANT project*. 2010. URL: `http://www.geant.net/`.

[57] Ian Horrocks et al. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission. World Wide Web Consortium, 2004. URL: `http://www.w3.org/Submission/SWRL`.

[58] Xin Jin, Ram Krishnan, and Ravi Sandhu. “Role and attribute based collaborative administration of intra-tenant cloud IaaS”. In: *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2014 International Conference on*. IEEE. 2014, pp. 261–274.

[59] Xin Jin, Ram Krishnan, and Ravi S Sandhu. “A Unified Attribute-Based Access Control Model Covering DAC, MAC and RBAC.” In: *DBSec* 12 (2012), pp. 41–55.

[60] Amit Sahai and Brent Waters. “Fuzzy identity-based encryption”. In: *Advances in Cryptology–EUROCRYPT 2005*. Springer, 2005, pp. 457–473.

[61] Vipul Goyal et al. “Attribute-based encryption for fine-grained access control of encrypted data”. In: *Proceedings of the 13th ACM conference on Computer and communications security*. ACM. 2006, pp. 89–98.

[62] John Bethencourt, Amit Sahai, and Brent Waters. “Ciphertext-policy attribute-based encryption”. In: *Security and Privacy, 2007. SP’07. IEEE Symposium on*. IEEE. 2007, pp. 321–334.

[63] Nigel P Smart and Frederik Vercauteren. “Fully homomorphic encryption with relatively small key and ciphertext sizes”. In: *Public Key Cryptography–PKC 2010*. Springer, 2010, pp. 420–443.

[64] Zvika Brakerski and Vinod Vaikuntanathan. “Efficient fully homomorphic encryption from (standard) LWE”. In: *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*. IEEE. 2011, pp. 97–106.

[65] Shucheng Yu et al. “Achieving secure, scalable, and fine-grained data access control in cloud computing”. In: *INFOCOM, 2010 Proceedings IEEE*. Ieee. 2010, pp. 1–9.

[66] Ming Li et al. “Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption”. In: *Parallel and Distributed Systems, IEEE Transactions on* 24.1 (2013), pp. 131–143.

[67] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. “Can homomorphic encryption be practical?” In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM. 2011, pp. 113–124.

[68] Quan Pham et al. “On a taxonomy of delegation”. In: *Computers & Security* 29.5 (2010), pp. 565–579.

[69] Jason Crampton and Hemanth Khambhammettu. “Delegation in role-based access control”. In: *Computer Security - ESORICS 2006*. Springer, 2006, pp. 174–191. ISBN: 978-3-540-44601-9.

[70] Tuan-Anh Nguyen et al. “Flexible and manageable delegation of authority in rbac”. In: *null*. IEEE. 2007, pp. 453–458.

[71] David W Chadwick, Sassa Otenko, and Tuan Anh Nguyen. “Adding support to XACML for multi-domain user to user dynamic delegation of authority”. In: *International Journal of Information Security* 8.2 (2009), pp. 137–152.

[72] Dave Cooper. “Internet X. 509 public key infrastructure certificate and certificate revocation list (CRL) profile”. In: (2008).

[73] Scott Cantor et al. “Assertions and protocols for the oasis security assertion markup language”. In: *OASIS Standard (March 2005)* (2005).

[74] Ed. D. Hardt and D. Recordon. *The OAuth 2.0 Authorization Framework, draft-ietf-oauth-v2-30*. Tech. rep. July 2012. URL: `http://tools.ietf.org/html/draft-ietf-oauth-v2`.

[75] Chang Jie Guo et al. “A framework for native multi-tenancy application development and management”. In: *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*. IEEE. 2007, pp. 551–558.

[76] Ravi Sandhu, Venkata Bhamidipati, and Qamar Munawer. "The ARBAC97 model for role-based administration of roles". In: *ACM Transactions on Information and System Security (TISSEC)* 2.1 (1999), pp. 105–135.

[77] 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.

[78] Vincent C Hu et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. Tech. rep. 2014, p. 162. URL: `http://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.sp.800-162.pdf`.

[79] Mattijs Ghijsen et al. "Towards an infrastructure description language for modeling computing infrastructures". In: *Parallel and Distributed Processing with Applications (ISPA), 2012 IEEE 10th International Symposium on*. IEEE. 2012, pp. 207–214.

[80] Joan A Garcia-Espin et al. "A multi-tenancy model based on resource capabilities and ownership for infrastructure management". In: *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE. 2012, pp. 682–686.

[81] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. "Securecloud: Towards a comprehensive security framework for cloud computing environments". In: *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*. IEEE. 2010, pp. 393–398.

[82] *Sun's XACML Implementation*. URL: `http://sunxacml.sourceforge.net/`.

[83] Canh Ngo, Yuri Demchenko, and Cees de Laat. "Decision Diagrams for XACML Policy Evaluation and Management". In: *Computers & Security* 49 (2015), pp. 1–16.

[84] OpenNebula. *OpenNebula Toolkit*. 2013. URL: `http://opennebula.org/`.

[85] *SNE-XACML Project, LGPL v3*. URL: `https://github.com/canhnt/sne-xacml`.

[86] *Apache ServiceMix v4.5.3*. `http://servicemix.apache.org/`. 2013.

[87] *Redis key-value data store*. 2013. URL: `http://redis.io/`.

[88] *Jena Semantic Web Framework, v2.11.0*. 2013. URL: `http://jena.apache.org/`.

[89] Canh Ngo, Y. Demchenko, and C. de Laat. "Toward a Dynamic Trust Establishment approach for multi-provider Intercloud environment". In: *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. 2012, pp. 532–538. DOI: `10.1109/CloudCom.2012.6427548`.

[90] Yuri Demchenko et al. "Intercloud Architecture for interoperability and integration". In: *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*. IEEE. 2012, pp. 666–674.

[91] Chin Guok et al. *On-Demand Infrastructure Services Provisioning Best Practices*. Tech. rep. OGF ISOD-RG, 2012.

[92] *XML Signature Syntax and Processing, 2nd. Ed.* 2008. URL: `http://www.w3.org/TR/xmldsig-core/`.

[93] *Apache Santuario project: XML Signature and Encryption Processing*. 2013. URL: `http://santuario.apache.org/`.

[94] *RFC3447: Public-Key Cryptography Standards (PKCS1): RSA Cryptography Specs v2.1*. 2003. URL: `http://tools.ietf.org/html/rfc3447`.

[95] *Bouncy Castle Java Crypto APIs v1.49*. `http://bouncycastle.org/java.html`. 2013.

[96] Fatih Turkmen and Bruno Crispo. "Performance evaluation of XACML PDP implementations". In: *Proceedings of the 2008 ACM workshop on Secure web services*. ACM. 2008, pp. 37–44.

[97] Yuri Demchenko, Canh Ngo, and Cees de Laat. "Access control infrastructure for on-demand provisioned virtualised infrastructure services". In: *Collaboration Technologies and Systems (CTS), 2011 International Conference on*. IEEE. 2011, pp. 466–475.

[98] Alex X Liu et al. "Designing fast and scalable xacml policy evaluation engines". In: *Computers, IEEE Transactions on* 60.12 (2011), pp. 1802–1817.

[99] Santiago Pina Ros, Mario Lischka, and Félix Gómez Mármol. "Graph-based XACML evaluation". In: *Proceedings of the 17th ACM symposium on Access Control Models and Technologies*. SACMAT '12. Newark, New Jersey, USA: ACM, 2012, pp. 83–92. ISBN: 978-1-4503-1295-0.

[100] Kathi Fisler et al. "Verification and change-impact analysis of access-control policies". In: *Proceedings of the 27th international conference on Software engineering*. ICSE '05. St. Louis, MO, USA: ACM, 2005, pp. 196–205. ISBN: 1-58113-963-2.

[101] Ninghui Li and Mahesh V Tripunitara. "Security analysis in role-based access control". In: *ACM Transactions on Information and System Security (TISSEC)* 9.4 (2006), pp. 391–420.

[102] Vladimir Kolovski, James Hendler, and Bijan Parsia. "Analyzing web access control policies". In: *Proceedings of the 16th international conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, 2007, pp. 677–686. ISBN: 978-1-59593-654-7.

[103] Hongxin Hu and GailJoon Ahn. "Enabling verification and conformance testing for access control model". In: *Proceedings of the 13th ACM symposium on Access control models and technologies*. ACM. 2008, pp. 195–204.

[104] Masahiro Fujita, Patrick C. McGeer, and JC-Y Yang. "Multi-terminal binary decision diagrams: An efficient data structure for matrix representation". In: *Formal methods in system design* 10.2-3 (1997), pp. 149–169.

[105] Massimiliano Masi, Rosario Pugliese, and Francesco Tiezzi. "Formalisation and implementation of the XACML access control mechanism". In: *Engineering Secure Software and Systems*. Springer, 2012, pp. 60–74.

[106] Piero Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. "An algebra for composing access control policies". In: *ACM Transactions on Information and System Security (TISSEC)* 5.1 (2002), pp. 1–35.

[107] P. Mazzoleni et al. "XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!" In: *Proceedings of the eleventh ACM symposium on Access control models and technologies*. SACMAT '06. Lake Tahoe, California, USA: ACM, 2006, pp. 219–227. ISBN: 1-59593-353-0.

[108] Glenn Bruns, Daniel S Dantas, and Michael Huth. "A simple and expressive semantic framework for policy composition in access control". In: *Proceedings of the 2007 ACM workshop on Formal methods in security engineering*. ACM. 2007, pp. 12–21.

[109] Qun Ni, Elisa Bertino, and Jorge Lobo. "D-algebra for composing access control policy decisions". In: *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*. ACM. 2009, pp. 298–309.

[110] Prathima Rao et al. "An algebra for fine-grained integration of XACML policies". In: *Proceedings of the 14th ACM symposium on Access control models and technologies*. ACM. 2009, pp. 63–72.

[111] Alex X Liu et al. "Xengine: a fast and scalable XACML policy evaluation engine". In: *ACM SIGMETRICS Performance Evaluation Review*. Vol. 36. ACM. 2008, pp. 265–276.

[112] Said Marouf et al. "Adaptive reordering and clustering-based framework for efficient XACML policy evaluation". In: *Services Computing, IEEE Transactions on* 4.4 (2011), pp. 300–313.

[113] Carroline Dewi Puspa Kencana Ramli, Hanne Riis Nielson, and Flemming Nielson. "The logic of XACML". In: *Science of Computer Programming* (2014), pp. 80 –105. ISSN: 0167-6423. DOI: 10.1016/j.scico.2013.05.003.

[114] Randal E Bryant. "Graph-based algorithms for boolean function manipulation". In: *Computers, IEEE Transactions on* 100.8 (1986), pp. 677–691.

[115] Karsten Strehl and Lothar Thiele. "Interval diagrams for efficient symbolic verification of process networks". In: *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 19.8 (2000), pp. 939–956.

[116] Mikkel Christiansen and Emmanuel Fleury. "An MTIDD Based Firewall". English. In: *Telecommunication Systems* 27.2-4 (2004), pp. 297–319. ISSN: 1018-4864. DOI: 10.1023/B:TELS.0000041013.23205.0f.

Publications

Publications in peer-reviewed journals

1. C. Ngo, Y. Demchenko, and C. de Laat, “Decision Diagrams for XACML Policy Evaluation and Management," In Computers & Security 49 (2015), pp. 1–16

2. C. Ngo, Y. Demchenko, and C. de Laat, “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services," in Journal of Information Security and Applications (*accepted 2015*).

Publications in peer-reviewed conference proceedings

1. C. Ngo, M. X. Makkes, Y. Demchenko, and C. de Laat, “Multi-data-types interval decision diagrams for XACML evaluation engine," in Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on, 2013, pp. 257–266.

2. C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures," in Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 343–349.

3. C. Ngo, Y. Demchenko, and C. de Laat, “Toward a Dynamic Trust Establishment Approach for Multi-provider Intercloud Environment," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012.

4. C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Security framework for virtualised infrastructure services provisioned on-demand," in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 698–704.

5. Y. Demchenko, C. Ngo, C. de Laat, D. R. Lopez, A. Morales, and J. A. Garcia-Espin, “Security Infrastructure for Dynamically Provisioned Cloud Infrastructure Services," in Privacy and Security for Cloud Computing, Springer London, 2013, pp. 167–210.

6. P. Membrey, K. C. Chan, C. Ngo, Y. Demchenko, and C. de Laat, “Trusted Virtual Infrastructure Bootstrapping for On Demand Services," in Availability,

Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 350–357.

7. Y. Demchenko, C. Ngo, P. Martinez-Julia, E. Torroglosa, M. Grammatikou, J. Jofre, S. Gheorghiu, J. A. Garcia-Espin, A. D. Perez-Morales, and C. de Laat, "GEMBus based services composition platform for cloud Paas," in Service-Oriented and Cloud Computing, Springer Berlin Heidelberg, 2012, pp. 32–47.

8. Y. Demchenko, C. Ngo, C. de Laat, T. W. Wlodarczyk, C. Rong, and W. Ziegler, "Security infrastructure for on-demand provisioned cloud infrastructure services," in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 255–263.

9. Y. Demchenko, C. Ngo, and C. de Laat, "Access control infrastructure for on-demand provisioned virtualised infrastructure services," in Collaboration Technologies and Systems (CTS), 2011 International Conference on, 2011, pp. 466–475.

Publication Authorship

Author contributions to the publications used in this thesis.

Chapter 2

- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Security framework for virtualised infrastructure services provisioned on-demand," in Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on, 2011, pp. 698–704 [48].

  C.N. designed, implemented and performed the experiments. P.M. contributed the secure bootstrapping protocol section. Y.D. consulted the study and publication. C.d.L supervised the work.

- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures," in Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 343–349 [49].

  C.N. designed, implemented and performed the experiments. P.M. contributed the bootstrapping trust management section. Y.D. consulted the study and publication. C.d.L supervised the work.

- C. Ngo, Y. Demchenko, and C. de Laat, “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services," in Journal of Information Security and Applications [50].

  C.N. designed, implemented and performed the experiments. Y.D. and C.d.L supervised the work.

Chapter 3

- C. Ngo, P. Membrey, Y. Demchenko, and C. de Laat, “Policy and context management in dynamically provisioned access control service for virtualized Cloud infrastructures," in Availability, Reliability and Security (ARES), 2012 Seventh International Conference on, 2012, pp. 343–349 [49].

  C.N. designed, implemented and performed the experiments. P.M. contributed the bootstrapping trust management section. Y.D. consulted the study and publication. C.d.L supervised the work.

- C. Ngo, Y. Demchenko, and C. de Laat, “Toward a Dynamic Trust Establishment Approach for Multi-provider Intercloud Environment," Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on. IEEE, 2012 [89].

  C.N. designed, implemented and performed the experiments. Y.D. consulted the study and publication. C.d.L supervised the work.

- C. Ngo, Y. Demchenko, and C. de Laat, “Multi-tenant Attribute-based Access Control for Cloud Infrastructure Services," in Journal of Information Security and Applications [50].

  C.N. designed, implemented and performed the experiments. Y.D. and C.d.L supervised the work.

Chapters 4 and 5

- C. Ngo, M. X. Makkes, Y. Demchenko, and C. de Laat, “Multi-data-types interval decision diagrams for XACML evaluation engine," in Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on, 2013, pp. 257–266 [47].

  C.N. designed, implemented and performed the experiments. M.X.M consulted the study and publication. Y.D. and C.d.L supervised the work.

- C. Ngo, Y. Demchenko, and C. de Laat, “Decision Diagrams for XACML Policy Evaluation and Management," In Computers & Security 49 (2015), pp. 1–16 [83].

  C.N. designed, implemented and performed the experiments. Y.D. and C.d.L supervised the work.

List of Figures

List of Tables

Summary

Access control is an important part of information security. It aims to preserve the confidentiality, integrity and availability by restricting access to protected resources and information via authorization. Depending on specific designs of computer systems, different access control models and mechanisms have been introduced.

The evolution of Cloud Computing brings advantages to both customers and service providers to utilize and manage computing and network resources more efficiently with virtualization, service-oriented architecture technologies, and automated on-demand resource provisioning. However, these advantages come with challenges on how to securely and efficiently protect customer resources in cloud environments. Service providers need to provide elastic and flexible cloud resources to their large numbers of customers based on the multi-tenancy model while ensuring reliable isolation on shared infrastructures. Therefore, designing and integrating access control mechanisms into cloud resource management is not trivial. Although many approaches have been proposed, they still suffer some drawbacks. First, they lack flexibility and interoperability with information models from management systems of on-demand provisioned cloud resources. Second, their policies and access control mechanisms are either coarse-grained, or do not have sufficient performance for large-scale cloud deployments.

This thesis contributes to the mentioned research field by investigating requirements of the access control for cloud infrastructure systems composed of compute and networking components. Based on these findings we propose a flexible and efficient access control approach that not only protects distributed cloud resources but also takes into account cloud infrastructure topology and characteristics.

Our work contains the following contributions:

We introduce a multi-tenant access control system with fine-grained authorization for cloud service management. It supports integration with the information model of cloud infrastructure management for providers. The proposed solution allows customers to dynamically create access control service instances together with policy definitions constrained in a SLA (Service Level Agreement) while deploying provisioned clouds. The approach supports on-demand provisioning and rescaling of cloud resources. It can regenerates policies to reflect changes in resource model descriptions. For Intercloud scenarios with clouds across multiple providers, we propose the authorization token exchange approach to solve distributed, inter-domain authorization and security context management problems. It allows users to established dynamic, fine-grained trust relationships with the chain of involved providers who may not have direct trust relations. The proposed solutions are implemented as a part of the GEYSERS project prototype and testbed. It demonstrates that our approach is flexible in supporting elastic resource scaling and re-planning scenarios. Experiments also prove that the performance of our prototype is acceptable for cloud providers with thousands of customers.

Moreover, to solve the bottleneck problems when using the XACML policy language in high performance authorization systems, we propose and implement a novel approach that includes modeling, analyzing and optimizing XACML policy elements. The proposed approach decomposes policies, aggregates and reduces the scattering of complex attribute criteria using interval processing mechanisms. It then constructs custom decision diagrams for XACML that increase efficiency of policy evaluation. We demonstrate our approach in our open source high performance policy evaluation engine developed for the XACML 3.0 standard. It not only achieves magnitudes of throughputs improvement comparing to previous work but also retains original XACML policy semantics and expressiveness.

Samenvatting

Toegangscontrole is een belangrijk onderdeel van de informatiebeveiliging. Het doel is om de vertrouwelijkheid, integriteit en beschikbaarheid te behouden door het authoriseren van toegang tot beveiligde bronnen en informatie. Afhankelijk van specifieke eigenschappen van computersystemen zijn in het verleden verschillende toegangscontrole modellen en authorisatie mechanismen geïntroduceerd.

De evolutie van Cloud Computing brengt efficientie voordelen voor zowel klanten en dienstverleners door het toepassen van virtualisatie, service-oriented architectuur technologieën en automatische op afroep beschikbaar maken van computer- en netwerk faciliteiten. Echter, deze voordelen komen met uitdagingen over hoe de data van de klant veilig en efficiënt te beschermen op cloud-omgevingen. Dienstverleners bieden elastische en flexibele cloud faciliteiten aan grote aantallen klanten en moeten daarom zorgen voor betrouwbare onderlinge isolatie op de onderliggende gemeenschappelijke infrastructuren. In het jargon heet dat: multi-tenancy. Het is niet triviaal om in zulke systemen toegangscontrole te ontwerpen en implementeren. Hoewel vele benaderingen zijn voorgesteld, lijden deze toch aan een aantal nadelen. Ten eerste hebben deze benaderingen niet de flexibiliteit en interoperabiliteit om met de informatie modellen van on-demand geleverde cloud resources management systemen om te gaan. Ten tweede zijn de policy beschrijvingen en toegangscontrole mechanismen niet voldoende fijnmazig of hebben niet voldoende capaciteit voor grote cloud-implementaties.

In deze dissertatie dragen we aan het genoemde gebied bij door onderzoek aan de eisen van de toegangscontrole voor cloud-infrastructuur. Wij stellen een flexibele en efficiënte toegangscontrole benadering voor dat niet alleen gedistribueerde cloud resources beschermt maar ook rekening houdt met cloud topologie en eigenschappen van de infrastructuur.

Ons werk bestaat uit de volgende bijdragen:

We introduceren een multi-provider toegangs controle systeem met fijnmazige authorisatie voor cloud service management. Dit systeem maakt integratie met het informatie model voor cloud beheer door aanbieders mogelijk. De voorgestelde oplossing stelt klanten in staat om controle op de toegang tot infrastructuur instanties dynamisch te creëren inclusief de policy definities gelimiteerd door de afgesproken SLA (Service Level Agreement) op het moment dat de cloud instantie actief gemaakt wordt. De aanpak ondersteunt het instant beschikbaar maken en herschalen van cloud instanties door regeneratie van policies als gevolg van veranderingen in de resource-model beschrijvingen. Voor Intercloud scenario's met

clouds over meerdere aanbieders verspreid, stellen wij de token gebaseerde autorisatie uitwisseling voor om gedistribueerde inter-domein autorisatie en veiligheidscontext problematiek op te lossen. Het stelt gebruikers in staat om dynamische, fijnkorrelige vertrouwensrelaties op te zetten met de keten van betrokken providers, die wellicht geen onderlinge vertrouwensrelaties hebben. De voorgestelde oplossingen zijn geïmplementeerd als een onderdeel van het GEYSERS project prototype en testbed. Hierbij is aangetoond dat onze aanpak flexibel is in de ondersteuning van elastische cloud aanbiedingen en herplanning scenario's. Experimenten toonden ook aan dat de prestaties van onze prototype acceptabel werkt voor cloud providers met duizenden klanten.

Daarenboven, om knelpunten op te lossen bij het gebruik van de XACML policytaal in hoge aantal transactie systemen, introduceren en implementeren wij een nieuwe benadering die het modelleren, analyseren en optimaliseren van XACML elementen omvat. De voorgestelde aanpak ontleedt policies, neemt aggregaties en vermindert de verstrooiing van complexe XACML attribute criteria door gebruikmaking van interval verwerking mechanismen. Via deze aanpak komen we dan met voor de actuele situatie specifieke beslissing diagrammen voor XACML die de efficiëntie van de policy evaluatie enorm verhogen. We presenteren een implementatie van onze open source policy evaluator gebaseerd op de XACML 3.0-standaard. We tonen aan dat onze aanpak in een high performance policy evaluatie module niet alleen enige orden van grootte performance verbetering geeft vergeleken met eerder werk, maar ook de originele XACML policy semantiek en expressiviteit behoudt.

Acknowledgements

Pursuing the PhD is a long journey in my life. When writing these lines, I am please that my PhD is almost accomplished. Actually, it would not be possible without help of many people that encourage, help and guide me along the way.

I would like to thank my promoter, Prof. Cees de Laat, for accepting me as a PhD student into the System and Network Engineering group and the GEYSERS research project. I would like to express my deepest gratitude for your endless guidance, support and valuable insights into my research directions.

I am truly thankful for my supervisor, Dr. Yuri Demchenko, for his support, continuous guidance and encouragements throught out my PhD. You patiently guide me into the research field, created opportunities to participate and work with people in the field. Without you I cannot accomplish this journey.

A special thanks to Prof. Pieter Adriaans for your critical questions and constructive advice in the field of mathematical logic to improve my thesis. I cannot finish my thesis without your kind support.

I thank all my colleagues in our research group for their intellectual dicussions and comments on my work: Paola, Cosmin, Mattijs, Marc, Ralph, Wibi, Daniel, Zhiming, Jeroen, Peter, Hao, Karel, Naod, Chariklis, Rudolf, Guido, Arie, Adam, Mikolaj, Reggie, Spiros, Fahime, Merijn, Ana Maria, Ana Lucia, Miroslav, Souley and Gerben. It was my pleasure to work, drink, hang out and play sports with you during four years at UvA. I also thank for people helping me on my teaching assistant work: Jaap, Niels and Toto.

I also thank for people in the GEYSERS project that I work with: Joan A., Jordi, Ester and Sergi at i2CAT, Spain; Shuping and Eduard at Essex, UK; Florian and Philip at SAP, Switzerland; Giada and Nicola at Nextworks, Italy; Jens at IBBT, Belgium; Damian, Krzysztof, Lukasz and Artur at PSNC, Poland and many others in the project. I have a chance to work with you in an international setting, where I can learn many useful things.

A special thanks to my colleagues at Hippo for your flexibility and support when I need to take some time off for my thesis completion. I also thank Arthur for helping me translate the summary to Dutch language.

I am grateful for all of my Vietnamese friends in the Netherlands for their friendship and for the great time we had in the last five years, from playing football, photography, catan, hanging around, travels, etc. All of them make my PhD life balanced and colorful.

Above all, I would like to thank my family: my parents, my brother and my

sister-in-law for their love and constant support. Huge thanks, love and appreciation go to my wife Van who was always there with grace and generosity. Thank our little son Kien who makes my life full of emotions and joys.

Canh Ngo
Amsterdam, January 2016.