An Ontology-Based Approach to Attribute Management in ABAC Environment

by

Ashwin Narayan Prabhu Verleker

A Thesis Presented in Partial Fulfillment
of the Requirement for the Degree
Master of Science

Approved April 2014 by the
Graduate Supervisory Committee:

Dr. Dijiang Huang, Chair
Dr. Gail-Joon Ahn
Dr. Partha Dasgupta

ARIZONA STATE UNIVERSITY

May 2014

ABSTRACT

Attribute Based Access Control (ABAC) mechanisms have been attracting a lot of interest from the research community in recent times. This is especially because of the flexibility and extensibility it provides by using attributes assigned to subjects as the basis for access control. ABAC enables an administrator of a server to enforce access policies on the data, services and other such resources fairly easily. It also accommodates new policies and changes to existing policies gracefully, thereby making it a potentially good mechanism for implementing access control in large systems, particularly in today's age of Cloud Computing.

However management of the attributes in ABAC environment is an area that has been little touched upon. Having a mechanism to allow multiple ABAC based systems to share data and resources can go a long way in making ABAC scalable. At the same time each system should be able to specify their own attribute sets independently .

In the research presented in this document a new mechanism is proposed that would enable users to share resources and data in a cloud environment using ABAC techniques in a distributed manner. The focus is mainly on decentralizing the access policy specifications for the shared data so that each data owner can specify the access policy independent of others. The concept of ontologies and semantic web is introduced in the ABAC paradigm that would help in giving a scalable structure to the attributes and also allow systems having different sets of attributes to communicate and share resources.

ACKNOWLEDGEMENTS

It is with immense gratitude that I acknowledge all the people who have inspired and helped me during my Master's studies.

First and foremost, I would like to thank my advisor, Dr. Dijiang Huang for his guidance during my study and research. It is his enthusiasm and persistence that gave me proper direction during my research. This thesis would not have been possible without his encouragement and sound advice.

My sincere thanks to my thesis committee; Dr. Gail-Joon Ahn and Dr. Partha Dasgupta, for taking time out from their extremely busy schedules and reviewing my work and giving valuable feedback.

I am extremely thankful to the fellow members of the SNAC research group at ASU; Bing Li, Zhijie Wang, Huijun Wu and Yuli Deng for all the help they provided during the course of the research work.

Last, but not the least, I would like to thank my family and friends who stood by me through all the good and bad times and who believed in me more than I did in myself. Any milestone achieved or to be achieved would never be possible without all these wonderful people in my life that I am blessed with.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Attribute Based Access Control

Attribute Based Access Control is a form of logical access control that includes access control lists, role-based access control and the ABAC method for providing access based on the evaluation of attributes [1]. Traditional access control mechanisms used an Access Control List (ACL) that would determine which user has what permissions on a particular resource. This Identity Based Access Control (IBAC) required the subject to present a credential that matches with the one held in the ACL in order to be given access to the requested object. Individual privileges of the subject to perform operations were managed on an individual basis by the object owner. Each object or resource needed its own ACL and a set of privileges assigned to each subject. The authorization decisions were made prior to any specific access request. This approach was fairly straightforward when the number of users and resources was less. But in today's age of large scale servers and clouds that span servers located in different geographical locations, this approach would be difficult to scale. For example, in a system having a million users, sharing a new resource would mean specifying which of these million users should get access to this resource. Therefore this approach was deemed cumbersome to manage and scale up based on the needs of the system.

The inherent problems in IBAC gave rise to another access control mechanism called Role Based Access Control (RBAC) [2] [3]. RBAC categorized users as satisfying particular roles defined by the system and multiple users could be grouped together based on the role which they have been assigned by the system. These roles are pre defined and carry a specific set of privileges associated with them. Users assigned to a role derive their privileges from it and access permissions are assigned to roles rather than individual users. The object or resource owner determines the roles that are given specific access privileges while sharing the resource and associates these permissions with each role. The major difference from IBAC is that now the role is considered to be a subject rather than an individual user where access control is concerned. At the point of an access request, the access control mechanism evaluates the role assigned to the subject that is requesting access and the set of operations this role is authorized to perform on the object before rendering and enforcing an access decision. RBAC made the management of access policies much easier and reduced the need to have an ACL associated with each resource.

Though RBAC solved many of the problems inherent in IBAC, it was still considered to be somewhat restrictive since the access decision was based on the role of the user and it was difficult to fit other factors in calculating the decision. As per [1] "both ACLs and RBAC are in some ways special cases of ABAC in terms of the attributes used. ACLs work on the attribute of 'identity' and RBAC works on the attribute of 'role'." ABAC is considered to be a more generic way of defining access policies and conditions since it can accommodate a broader set of factors in defining the policies than the other mechanisms of access control. An access policy in ABAC can express a complex boolean rule set that can evaluate a variety of attributes in a much easier manner as compared to IBAC or RBAC. It is also easier to update access rules in ABAC since it is much more difficult to identify all the places where the ACL or RBAC implementation needs to be updated [1].

ABAC is advantageous especially in a large enterprise environment where maintaining an access list for users or a list of roles for the users can be a very tedious task.

Today's large scale computation environments need to accommodate a large number of users (potentially infinite) and continuous changes in the operating conditions. An access control mechanism for such an environment needs to be highly flexible in order to gracefully manage the large user base as well as gracefully manage the frequent changes without affecting the performance of the system and introducing unwanted delays. ABAC avoids the need for explicit authorizations to be directly assigned to individual subjects prior to a request to perform an operation on an object. It is the natural convergence of existing access control models and surpasses their functionality [4]. Moreover, it enables flexibility by leveraging consistently defined attributes that span both subjects and objects. It allows for other activities like authentication and authorization in the same or different infrastructures while maintaining a proper level of security.

1.2 Ontologies and Semantic Web

The concept of ontologies was introduced in Computer Science because of the need for knowledge representation in artificial intelligence and other fields of informatics. If we consider two articles that describe the same concept, the words and language used to describe in both articles will be different and therefore it is difficult to apply machine learning techniques to analyse the data. Thus representing the meaning of concepts rather that just giving computer instructions is increasingly important. This is where the concept of ontologies comes into picture.

Gruber [5] defined ontologies as an "explicit specification of conceptualization". He considers conceptualization to be "an abstract, simplified view of the world that we wish to represent for some purpose". Ontologies are a way to share knowledge between different components of a system like two programs that work on similar data. They are a form of agreement or contract between all the components regarding the "meaning" of the concepts and data that they work with. If we consider a system that describes various sports for example, one component can describe "Table-tennis" and another component can call it "Ping-pong". An ontology can define an agreement that say that both "Table-tennis" and "Ping-pong" in fact refer to the same sport. Such an agreement can allow different component to use different terms to define the same concept, which in this case is the sport referred to by "Table-tennis" and "Ping-pong". Such a system allows distributed and independent development of system components as long as they abide by the "agreement" that is defined by the system. This can be considered similar to the use of interfaces in the Java programming language where different classes can be used for the same task (done in different ways by each class) as long as they implement the defined interface.

An important concept in the world of semantic web is the **Resource Description Framework (RDF)** [6]. It is a standard created by the Worldwide Web Consortium (W3C) with a goal to "enable and promote encoding, exchange and reuse of structured metadata". It is expressed using the eXtensible Markup Language (XML) with added constraints. The **Web Ontology Language (OWL)** [7] adds semantic meaning to the data expressed in RDF and is ideal for creating ontologies. According to the W3C specification for OWL [7] "OWL's ability to express ontological information about individuals appearing in multiple documents supports linking of data from diverse sources in a principled way." This allows the ontology specification to be performed in a distributed manner and express equivalences and hierarchies that can state the relation(s) between individuals or properties that have been defined by different attribute authorities. The information about the different attributes can be merged and this aggregation can be used to determine facts that are not directly represented in any one source [7].

1.3 Attribute Based Encryption

Attribute based encryption was first proposed in 2005 by Sahai and Waters as a cryptographic scheme to provide anonymous access control [8]. Each ciphertext is associated with a policy that determines which user or class of users can access or decrypt the encrypted data. It is a modified version of the traditional public key encryption scheme and ABE provides multiple private keys corresponding to a public key. Each private key has associated with it a set of attributes and their corresponding values that define the access permissions of the user to whom the key has been issued. When the user attempts to decrypt the ciphertext using his/her private key, the ABE algorithm checks whether the user's attributes (which are associated with the private key) match the encryption policy of the ciphertext. Only if the algorithm finds a match, the text or data is decrypted successfully.

Basically ABE comes in two flavours namely ciphertext-policy ABE (CP-ABE) and key-policy ABE (KP-ABE). Both these are explained later in the document in the detailed section on ABE. Though ABE is still not in widespread use due to slower performance as compared to traditional symmetric and asymmetric encryption schemes, it still has a big potential to enforce coarse as well fine grained access control policies with the same ease. The main advantage of using ABE is that key management is more simple from the user perspective. If we consider a scenario that a user wants to share a file with n other users. In the traditional public key cryptography scenario, the user would need to know the public keys of each of the n users, encrypt the file with each of these n public keys and then send the encrypted files to their intended recipients. Contrastingly, using ABE, the user simply needs to encrypt it once using the public key and then specify the access control policy that would define this set of n users.

The policy is generally a boolean expression over a set of attributes. If we consider the system to be that of a university and we need to share some data only with say, the faculty of Computer Science department. A sample policy that would define this set would be

(Faculty $\wedge$ CS dept)

ABE can support much more complex attribute policies where specific values or range of values can be specified by the user for the attributes using boolean relation operators. This support makes it very easy for the resource owner to specify the target set of users based on their attributes and does not need to know details that are specific to an individual user. Addition of new users is also fairly straightforward since the new user only needs to be provided with required attributes without changing the ciphertext access policy. This allows the resource owner to encrypt the resource just once (unless of course, the access policy needs to be changed).

Since the main feature of ABE is the use of attributes for specifying access permissions and rules, it requires a trusted authority to decide the attributes to be used in the system and also to issue private keys to the users. This Attribute Authority (AA) possesses a secret master key that is used to generate the private keys to be issued to the users. ABE provides a setup function to generate a master key and a corresponding public key that would be used by the authority in generating the private keys. These private keys are issued to the users when they register with the system and the keys ensure that the users get access to only those resources for which they are authorized. The security and reliability of ABE is dependent on the trustworthiness of the AA. If the AA is compromised or if the master key is stolen then it is easy to break the security protocol.

There have been previous attempts to integrate ABE with access control mechanisms like [9], [10] and [11]. The research presented in this document also tries to use

ABE, particularly CP-ABE to provide additional security over the ABAC mechanism that is introduced.

1.4 How does it come together?

Now that three fairly different and independent topics have been introduced, a fundamental question comes to mind that how can we use them together to design a robust system that is secure and at the same time easy to use. It is easy to understand how we can merge ABAC and ABE techniques since both are somewhat related due the use of attributes to define access policies. ABE techniques can be used to provide an additional layer of security to the data and resource that users share with each other in a cloud, for example so that even if a user manages to forge the access credentials of another user at the ABAC layer, he would get the resource in an encrypted format. Until and unless the user has access to the forged user's private key, he won't be able to decrypt the resource. Since private keys are generally secured outside the system with possibly physical security too, we can consider ABE to be secure in this scenario.

Now that it is somewhat established that ABAC and ABE would possibly work together, the next problem that arises is how does a user specify an access policy. The first technique that comes to mind is let the server or administrator do the work of generating the policy thereby providing a centralized way of ensuring coherence between multiple access policies. The server can then do the job of encrypting the resource before sharing it with the intended set of users. This works for most cases, however ABE, in its current state is computationally more intensive than other established symmetric and asymmetric cryptographic techniques. In a cloud, with possibly millions of users this design could possibly overload the server's computational resources thereby impacting performance negatively. This is where I introduce the use of ontologies to de-centralize the entire process to a large extent and reduce the load on the server while ensuring coherence of attributes.

1.5 Research Outline

In the following sections I describe in more detail the architecture and processes of my proposed design.

I first give a comprehensive overview of the Attribute Based Access Control model in Chapter 2.

Chapter 3 will concentrate on Ontologies and semantic web and discuss how giving a semantic structure to data helps improve the performance and complexity of our systems.

Chapter 4 will briefly cover the concept of Attribute Based Encryption with special focus on its potential as a possible encryption mechanism in real world situations.

Chapter 5 will bring it all together and we will see how Ontologies can be used for management of user attributes in a distributed ABAC environment along with using ABE for cryptographic purposes.

Finally in Chapter 6 I will discuss the current open problems and suggestions for future work in this area.

Chapter 2

ATTRIBUTE BASED ACCESS CONTROL

As per the ABAC document released by NIST in 2014 [1], "The concept of ABAC represents a point in the space of logical access control that includes access control lists, role-based access control, and the ABAC method for providing access based on the evaluation of attributes." ABAC expands the concept of Role-based Access Control (RBAC) to allow for a broader set of attributes to be used in addition to the "role" that a user is allotted in the system. In a sense, we can think of RBAC as a subset of ABAC.

The following is a high-level definition of ABAC [1]

ABAC is an access control method where subject requests to perform operations of objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions

Attributes are characteristics of the subject, object, or environment conditions. Attributes contain information given by a name-value pair.

A **subject** is a human user or NPE, such as a device that issues access requests to perform operations on objects. Subjects are assigned one or more attributes and are synonymous with users.

An **object** is a system resource for which access is managed by the ABAC system. It can be the resource or requested entity, as well as anything upon which an operation may be performed by a subject including data, applications, services, devices, and networks.

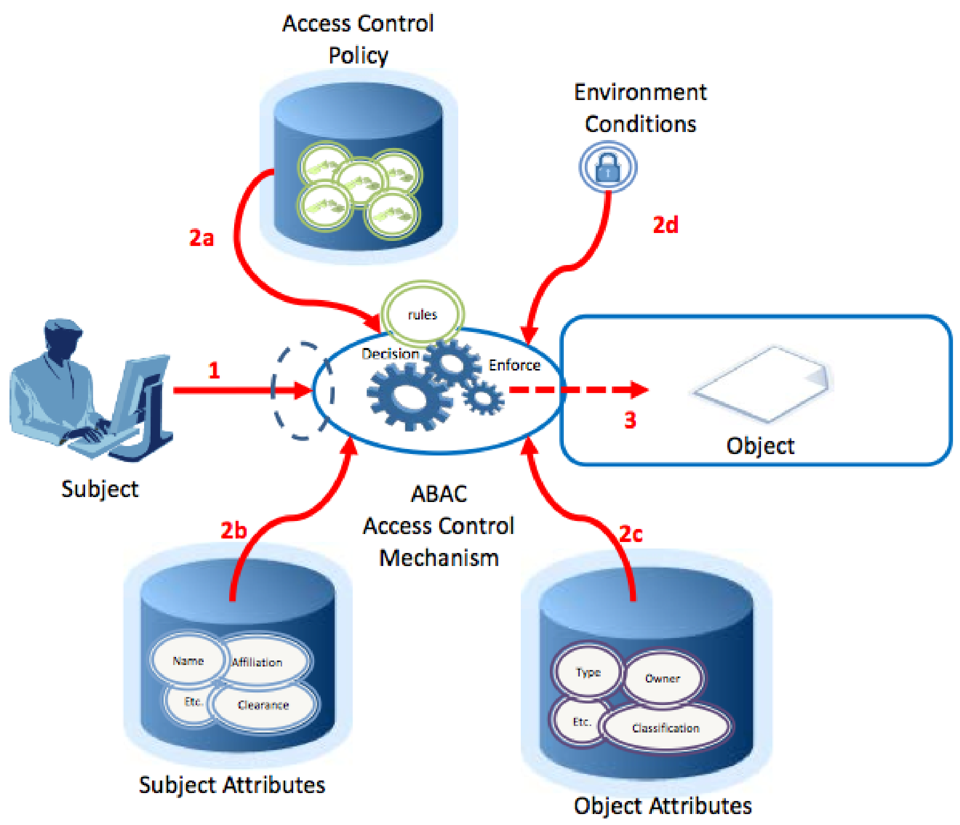


Figure 2.1: Basic ABAC Scenario [1]

An **operation** is the execution of a function at the request of a subject upon an object. Operations include read, write, edit, delete, copy, execute, and modify.

Policy is the representation of rules or relationships that makes it possible to determine if a requested access should be allowed, given the values of the attributes of the subject, object, and possibly environment conditions.

Environment conditions are operational or situational context in which access requests occur and these are independent of the subject or object.

An ABAC system assigns attributes to individual users that help in defining the access permissions to the users. Whenever a user requests access to a particular resource, the system will check if the user's attributes satisfy the access policy defined for that resource. Only if the user attributes satisfy the policy is the access provided to the user. Every time a new user (or subject) is created, it needs to be assigned its set of attributes. A change in the access permission level for a subject would basically mean a change in the attributes for that subject. The establishment of a user as a subject and specification of the attributes is done by a trusted administrator. These attributes are meant to capture the user and system level characteristics of that user. An attribute authority within the system maintains the subject identity information for the resource management system. The subject attributes may need to be updated every time new users are added or old users are removed. As new subjects join the organizations, rules and objects do not need to be modified. As long as the subject is assigned the attributes necessary for access to the required objects, no modifications to existing rules or object attributes are required. This is especially advantageous when we use ABE for encrypting the resources using attribute policies, as the object need not be encrypted again (encryption is considered to be a computationally intensive operation).

Every resource or object in the system that is intended to be shared securely among the different subjects, needs to have an access policy defined for it. This access policy is usually a boolean expression over the set of attributes defined by the system for the subjects, operations and environment conditions. As per [1] , ”this policy is normally derived from documented or procedural rules that describe the business processes and allowable actions within the organization”. A good example would be the case of healthcare systems where only a limited number of authorized users are allowed to access a patients medical records. The more sensitive the data, the more strict the access policy is.

2.1 Performance Requirements of ABAC

ABAC is mainly intended to simplify access control in enterprise environment where different organizations of the same enterprise manage resources in a distributed manner and one organization can have access to authorized resources managed by another organization. This interaction can be quite complex since the components are usually distributed across networks and geographical locations. The larger and more diverse the enterprise, the more complex the interactions get. One request to access a resource can lead to requests for attribute evaluation, policy checking, validation of request, integrity of resource, etc and thus the benefit of implementing ABAC mechanisms need to be carefully evaluated before the decision is taken. The scalability of the ABAC system is one of the most important aspects that needs to be considered. In an enterprise environment, it should be easy to add more organisations and remove existing organizations without disturbing the existing set up. In the context of ABAC, the attribute management is an area to be considered since the allowed sets of attributes can change over time especially when organizational restructuring occurs. When implementing ABAC over an existing system, the distribution of the ABAC components needs to take into account the existing underlying architecture and accordingly determine an efficient manner to store and share the data with minimum modification to the existing system. While designing an ABAC system for an enterprise, one needs to consider the amount of coupling that has to occur with the existing software architecture. A loosely coupled system would be more beneficial for maintainence as the ABAC mechanism can be maintained independently of the other components thereby reducing the maintenance costs involved. In the context of extensibility, the ABAC architecture needs to be flexible enough to allow for additional security mechanisms to be added to it or above it. It allows the customization of the mechanism as per the enterprise requirements and does not restrict the architecture to one particular implementation.

So What's the Problem?

In the current era of large scale enterprise systems and cloud computing, ABAC can be a strong mechanism to implement access control. So far we have discussed the design of an ABAC system with a single attribute authority that takes care of the attribute assignment to the subjects and the specification of access policies on the objects. In a large scale system, the question of scalability comes up. So can we scale ABAC with its current structure of a single attribute authority? This is the main issue that I aim to resolve in my research work and this is where I intend to introduce the concept of Ontologies (described in Chapter 2) into the ABAC system design.

In the scalable ABAC design, we have multiple "systems" that interact with each other and share data and resources. Each individual system has its own attribute authority that is responsible for the attribute management within its own domain. Each attribute authority independently defines its own set of attributes to be used by the users within its domain and these can be similar or different from those defined by other attribute authorities. Now when these multiple system need to share data and resources, we need to define access policies that would be valid for the users of all the different systems. The main problem in this scenario is how do we negotiate the attributes between two different systems so that the access policy of one system can be used by the other system as well.

When the different systems decide to share their resources, an initial negotiation of attributes needs to take place. A common Ontology is established, that would be used by all the systems to define their access policies. The advantage of using an ontology would be that it can define equivalence and hierarchy among different attributes. For example, the ontology can say that an attribute *attr1* from system *S1* is equivalent to attribute *attr2* from system *S2*. Therefore, if an object from *S2* is shared with access policy *attr2*, a user in *S1* having attribute *attr1* would be able to access it. A more concrete example would be in an academic environment where a project is being done in collaboration between two institutions. One institution specifies that certain data is to be accessed only by users having *Faculty* attribute. However the second institution does not have *Faculty* as a part of its attribute set. Therefore without an ontology, a user from the second institution will not be able to access that particular data. Now if we define a common ontology that says that the attribute *Faculty* is equivalent to the attribute *Teacher* (which is a part of the attribute set of the second institution), the concerned attribute authority can translate the access policy in a form suitable for the user of the second institution and thus any user having the attribute *Teacher* will also be able to access the data meant for users with attribute *Faculty.*

So far, this looks to be a good possible solution to allow for a distributed approach to specifying access policies in an ABAC environment. How exactly we can define equivalent attributes will be described in the next chapter where Ontologies will be described in detail.

Chapter 3

ONTOLOGIES AND SEMANTIC WEB

In this chapter I will give an overview of the concept of Ontologies and the Semantic web and focus more on how it can be used to allow for flexible management of attributes in a distributed ABAC environment.

Ontologies are a formal representation of knowledge in the form of sets of concepts in a domain and denotes the types, properties and relationships between the concepts. These have been extensively used to organize information in various fields like artificial intelligence, biomedical informatics, library science, etc. They are a way to share knowledge between different components of a system, such as two programs that work on similar data. They form an agreement or contract between the components regarding the ”meaning” of the concepts or data that they work with.

3.1 OWL basics and SPARQL

The standardized form of expressing ontologies is the Web Ontology Language (OWL) [7] which adds semantic data to ontologies expressed using the Resource Description Framework (RDF). RDF is a standard created by the Worldwide Web Consortium (W3C) that was developed with the goal to enable and promote encoding, exchange and reuse of structured metadata [6]. RDF is commonly expressed in the eXtensible Markup Language (XML) with some added constraints to express the semantics of the meta data. A sample ontology using OWL/RDF is given below.

```
<rdf:RDF xmlns="http://www.w3.org/2002/07/owl#"
     xml:base="http://www.w3.org/2002/07/owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
```

```
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<Ontology rdf:about="http://www.semanticweb.org/anprabhu/ontologies/2014/2/
    ASU_User"/>

<DatatypeProperty rdf:about="http://www.semanticweb.org/anprabhu/ontologies
    /2014/2/ASU_User#age"/>
<DatatypeProperty rdf:about="http://www.semanticweb.org/anprabhu/ontologies
    /2014/2/ASU_User#department"/>
<DatatypeProperty rdf:about="http://www.semanticweb.org/anprabhu/ontologies
    /2014/2/ASU_User#level"/>
<DatatypeProperty rdf:about="http://www.semanticweb.org/anprabhu/ontologies
    /2014/2/ASU_User#name"/>
<DatatypeProperty rdf:about="http://www.semanticweb.org/anprabhu/ontologies
    /2014/2/ASU_User#title"/>

    <Class rdf:about="http://www.semanticweb.org/anprabhu/ontologies/2014/2/
        ASU_User#Employee">
    <equivalentClass rdf:resource="http://www.semanticweb.org/anprabhu/ontologies
        /2014/2/ASU_User#User"/>
</Class>

<Class rdf:about="http://www.semanticweb.org/anprabhu/ontologies/2014/2/ASU_User#
    Member">
    <equivalentClass rdf:resource="http://www.semanticweb.org/anprabhu/ontologies
        /2014/2/ASU_User#User"/>
</Class>

<Class rdf:about="http://www.semanticweb.org/anprabhu/ontologies/2014/2/ASU_User#
    User"/>
```

This ontology follows the OWL syntax. Note that this is just one of the multiple syntaxes that are supported in ontologies. Every ontology defines a *namespace* for all the classes, properties and individuals that it defines. This allows multiple ontologies to use the same name for its own declarations provided the namespaces are different and promotes re-usability of names. Different prefixes are also declared to allow the usage of names and values defined in other namespaces. This ontology defines three classes viz. *Employee, User, Member.* These classes are declared as equivalent, which means that an individual, which is an instance of class Member, is also considered to be a valid instance of the other two classes and vice versa. A few *DataProperties* are also declared and these are the properties to be assigned to the individuals in the ontology. In the context of ABAC, these properties would be the attribute names and their corresponding values would represent the values for those attributes for the user. Similar to equivalent classes, we can also have equivalent properties (not shown in the example above) that would allow different attribute names to correspond to the same attribute.

SPARQL (recursive acronym for SPARQL Protocol and RDF Query Language) [12] is a query language designed to be used with OWL/RDF ontologies. As per the W3C specification [12], SPARQL is capable of querying required and optional graph patterns along with their conjunctions and disjunctions. It also supports extensible value testing and constraining queries by source RDF graph. The syntax of SPARQL is similar to the commonly known SQL query language for databases, thereby making it easy to learn and use. A sample SPARQL query is shown below.

```
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 asu:<http://www.semanticweb.org/anprabhu/ontologies/2014/2/ASU_User#>
SELECT ?x
        WHERE { ?x rdf:type asu:Employee .
                ?x asu:level ?level .
                FILTER (STR(?level) = "manager")
        }
```

This simple SPARQL query selects individuals from the ontology which are instances of the *Employee* class whose *level* is ”manager”. We define variables using the ? prefix to the variable name. In this example, the variables are ?x and ?level.

The rdf:type refers to the class of the individual that is being searched (in this case the class is *Employee* from the namespace defined by the asu prefix. The results are filtered based on the value of the *level* data property of the *Employee* individual and only those individuals that satisfy the filter condition are returned. One of the advantage of SPARQL is it will automatically consider the equivalence of classes and properties while executing the query. For example, if we consider the ontology structure defined earlier, this query will also return any individuals of type *Member* and *User* that have the *level* property value "manager".

A lot of tools are publicly available, many of them open-source which help you to create and edit ontologies in a very user friendly way. In my implementation, I have used the Protege Ontology Editor [13]. It is an open source software that can create, validate and edit ontologies and also provides interface to run SPARQL queries. Many tutorials like [14] are available that show how to use this software.

In the implementation (described later in the document), I would be using SPARQL to define the access policies for the shared resources. This choice is made because it would give the system an easy way of handling the ontology and attribute properties in a reliable manner. SPARQL, combined with the Pellet Reasoner for OWL ontologies [15] is a very handy tool for querying and validating ontologies.

3.2 Ontologies in ABAC

Now that we are familiar with ABAC and Ontologies, let us see how they can work together. Recalling the attribute assignment in ABAC, the Attribute Authority is responsible for the creation and assignment of attributes to users and other subjects. The aim of using ontologies is to allow for sharing of data between systems that use different sets of attributes without having to translate access policies every time. In the proposed design of this thesis, all the various AAs initially decide their own sets of attributes to be used in the system under them. There occurs a negotiation of these different attribute sets between the AAs and all the equivalent attributes are defined. All these attributes are then given a proper structure and definition using an ontology that would specify the equivalence of all the different attributes.

Now that we have an attribute structure (ontology) that is agreed upon by all the concerned attribute authorities, it is fairly straightforward for an AA to translate and evaluate an access policy that is defined by a subject from another ABAC system. The ontology enables an AA to create an equivalent policy for its own system by replacing the attributes in the access policy (if required) by their equivalent attributes in its system. If certain attributes are not present in the AAs domain, they can be ignored as they are not relevant in that particular domain. To illustrate, consider the attribute set A = $\{attr_1, attr_2, attr_3, attr_4\}$ used by attribute authority Auth. Let A' = $\{attr'_1, attr'_2, attr'_3, attr'_4\}$ be the attribute set used by attribute authority Auth'. A user from the domain of Auth shares a file with access policy as $attr_1 \wedge (attr_2 \vee attr_3)$ and wants to share it with users from domain of Auth'. If the ontology defines that $attr_i \equiv attr'_i$, then Auth' can translate the access policy to $attr'_1 \wedge (attr'_2 \vee attr'_3)$ which would be a valid access policy for any user in the domain of Auth'.

If we consider the same scenario without a defined ontology, Auth and Auth' would need to have a negotiation first to map their attributes to one another for Auth' to create its equivalent policy. The attribute sets in this example have four attributes each, however in a real-world situation there can be potentially a very large number of attributes in a system depending on its security and access requirements. Mapping of attributes every time some data needs to be shared will prove to be computationally intensive and thus not feasible.

In Figure 3.1 we see the overall architecture of the proposed design. Each AA

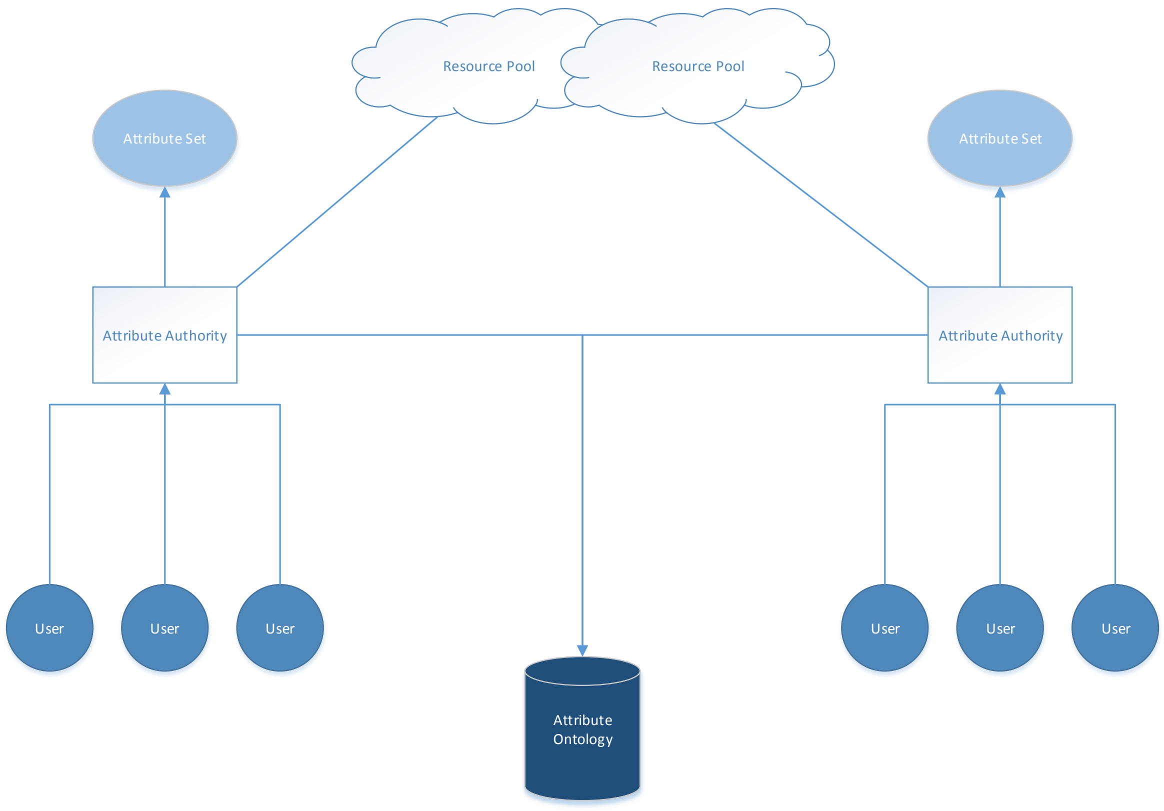


Figure 3.1: Overall Architecture

represents an independent ABAC system and has its own attribute set to be used within that system. Multiple users are present in all the ABAC systems and there is a pool of resources that are shared securely by the users within it. Some of these resources may be shared with other systems (represented by the intersection of the resource pools). There is a common attribute ontology that is set up initially and it defines the mapping between the attributes of the multiple ABAC systems. All the AAs involved can access the ontology and translate the access policies defined by another system to accommodate its own users in the shared pool of resources.

Advantages

- Attribute Authorities need not negotiate every time they needs to share data with each other.
- The same access policy can be used for sharing in any ABAC system that is a part of the ontology.
- New AAs can join the ontology by adding their own attributes into the already mapped sets of the ontology. This way, all the AAs need not negotiate each time a new AA is added to the ontology.
- It allows for selective use of attributes in a shared environment. For example, if a particular AA wants to restrict which users are to be allowed to share resources in the common pool, it can add only those attributes in the ontology.
- If ABE is used to encrypt the data (described in the next chapter), having an ontology means that the data need not be encrypted or decrypted again to allow for different attribute names. This is a big saving where computation time and resources are concerned.
- In an Information Centric Network (ICN) environment, this allows for a single copy of the data to be present in the network cache, thereby saving a lot on the limited memory space.

Chapter 4

CIPHERTEXT-POLICY ATTRIBUTE BASED ENCRYPTION

Ciphertext-Policy Attribute Based Encryption (CP-ABE) was introduced by J. Bethencourt *et al* [16]. In this scheme, each user is assigned a set of attributes that represent the roles and access permissions of that user in the system. Each user is assigned a private key that has the attributes issued to the user and this can be used to decrypt the encrypted data that the user tries to access.

In a sense CP-ABE is similar to the Key-Policy ABE (KP-ABE) scheme described in [17], however it has significant differences as far as the construction is concerned. In KP-ABE, ciphertexts are associated with a set of descriptive attributes, and user's keys are associated with policies. This is exactly the reverse of the situation in CP-ABE. As argued in [16], in KP-ABE, the encryptor exerts no control over who has access to the data being encrypted, except by the choice of the descriptive attributes for the data. The encryptor must trust that the key-issuer issues the appropriate keys to grant or deny access to the appropriate users. In contrast, CP-ABE attaches the policy to the ciphertext and the descriptive attributes to the private keys. This gives the encryptor the power and flexibility to explicitly specify the access policy to the encrypted data being shared. For example, ABE can be used to encrypt a file using the policy

$Professor$ **and** (CS_Dept **or** EE_Dept)

If a user obtains a private key with an assignment of attributes that satisfies either of the two allowed combinations as per the above policy, he/she can decrypt the file based on the match between access policies embedded into the file and identity attributes described in the user's private key.

The motivation behind using ABE in an ABAC system is that it lends itself easily to the ABAC concept since both are based on sets of descriptive attributes and resource access based on policies and attribute assignment based on these attribute sets. In addition to specifying an access policy from the ABAC perspective, the user can use that policy specification to encrypt the data considering that the attribute sets in ABAC and ABE is the same. This adds an additional layer of security to ABAC. In case a user's credentials are compromised, the attacker may be still able to download the file/resource. However with the use of ABE, the resource will be in the encrypted form and the attacker will not be able to decrypt it unless he has access to the private key of the user whose credentials have been compromised.

The CP-ABE algorithms are described in Appendix A. This chapter will focus more on how CP-ABE can be used in the ABAC environment that we have set up so far. For the sake of simplicity, wherever mentioned, encryption/decryption is synonymous with CP-ABE encryption/decryption in the rest of the document unless explicitly mentioned otherwise.

As mentioned earlier, the user's private keys (or decryption keys) are associated with a set of descriptive attributes. In our ABAC environment, these attributes will be specified by the Attribute Authority. For string attributes, CP-ABE uses string literals as attributes for encryption and decryption. If we use CP-ABE in its pure form, then it will not be possible to allow for attribute equivalence. For example, if a subject in the ABAC system has attribute *User*, and an encrypted document had used *Member* as the encryption policy. As per our ontology based design described in the previous chapter, the user would be allowed access to the document (provided *User* and *Member* have been declared as equivalent attributes in the ontology). Since CP-ABE does not currently support such an equivalence of attributes, we modify our design slightly to allow for CP-ABE to be used along with attribute equivalence.

As seen in the previous chapter, we can define equivalent attributes in the OWL ontology. We now go a step further and map all equivalent attributes to a single *attribute-id*. All equivalent attributes have the same *attribute-id*. While creating the policy for cryptographic purposes, we use this *attribute-id* to encrypt the data instead of using one of the attributes from the equivalent set. The advantage of using this approach is that the encryption policy does not change for different ABAC systems as all the equivalent attributes will be mapping to the common *attribute-id*. Even when the decryption keys are issued to the subjects, the *attribute-ids* are used to specify the subject's attributes in the private key.

To implement this, we add an *Object Property* named "hasUniqueName" to the ontology. A new class is created for every unique name required and the corresponding equivalent classes are mapped to the unique name class using the hasUniqueName property. In my implementation, I am using the "hasUniqueName some UniqueClass" cardinality and it can vary across implementations. This just means that the target of the hasUniqueName object property would be the UniqueClass class. All the equivalent attributes are mapped to the same unique class via this property. An example where a class is mapped using the hasUniqueName property is shown below.

```
<owl:ObjectProperty rdf:about="&OntologyABAC_Demo;hasUniqueName"/>

<owl:Class rdf:about="&OntologyABAC_Demo;Junior">
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty rdf:resource="&OntologyABAC_Demo;hasUniqueName"/>
                <owl:someValuesFrom rdf:resource="&OntologyABAC_Demo;UniqueJunior"/>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

In this case, the class *Junior* is mapped to class *UniqueJunior* via the hasUniqueName object property. Any other class in the ontology that is declared to be equivalent to *Junior* will also need to be mapped to the *UniqueJunior* class in a similar fashion. With this defined, whenever we want the unique attribute/class name to use in the ABE encryption policy, we can query the ontology for the range of the ”hasUniqueName” object property. The SPARQL query for achieving the purpose for this example is given below.

```
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 asu: <http://www.semanticweb.org/ontologies/2014/2/OntologyABAC_Demo.owl#>
SELECT ?x
WHERE
{
        asu:className rdfs:subClassOf ?object .
        ?object a owl:Restriction .
        ?object owl:onProperty asu:hasUniqueName .
        ?object owl:someValuesFrom ?x
}
```

This query again assumes that the ”some” cardinality has been used in specifying the hasUniqueName object property. It can change depending upon the implementation. Basically this query takes a ”className”, which is the domain class that the user is looking to find the unique name for, and then detects the target of the object property for that class. This target is then stored in the variable x and that is displayed/returned. In the implementation, if a class does not have the object property defined, we can assume that it does not have other equivalent classes and therefore the class name itself can be used as an attribute for encryption purposes.

In this manner, whenever we need the unique name for any attribute, we simply need to query for the range of the object property giving the domain as the attribute for which the unique name is being searched. This approach ensures that all the data is being encrypted by using the unique names as attributes in the encryption policy.

The same names are used to specify attributes in the decryption keys of the users. This ensures that the encryption and decryption is coherent and any valid user can decrypt the data that is encrypted by a user of another ABAC system.

Chapter 5

IMPLEMENTATION AND DEMONSTRATION

In this section I will describe my implementation of the proposed design. Figure 5.1 describes the workflow of the implemented system.

The establishment of the ontology is assumed to be done when the system is set up initially. All the n ABAC systems share their own individual attribute sets and negotiate in order to come up with an agreed-upon ontology. This ontology is then used by the users to create their access policies and share the data. Below is a summary of the tools I have used in the implementation.

Table 5.1: Summary of Software Tools Used

Tool Name	**Purpose**
J2EE	Server Side Development
Apache Tomcat 6	Web Server
HTML, CSS	User Interface
Protege 4.3	View and Create Ontology
Apache Jena	Ontology library
Pellet Reasoner	SPARQL query execution

The software has been implemented as a web application that allows users to share files among themselves. The admin can upload an ontology to the server which all the users can access. A user can upload files to the server, however these files cannot be downloaded until the owner of the file decides to share it. When the owner shares a particular file, he has to upload a corresponding access policy. The server checks

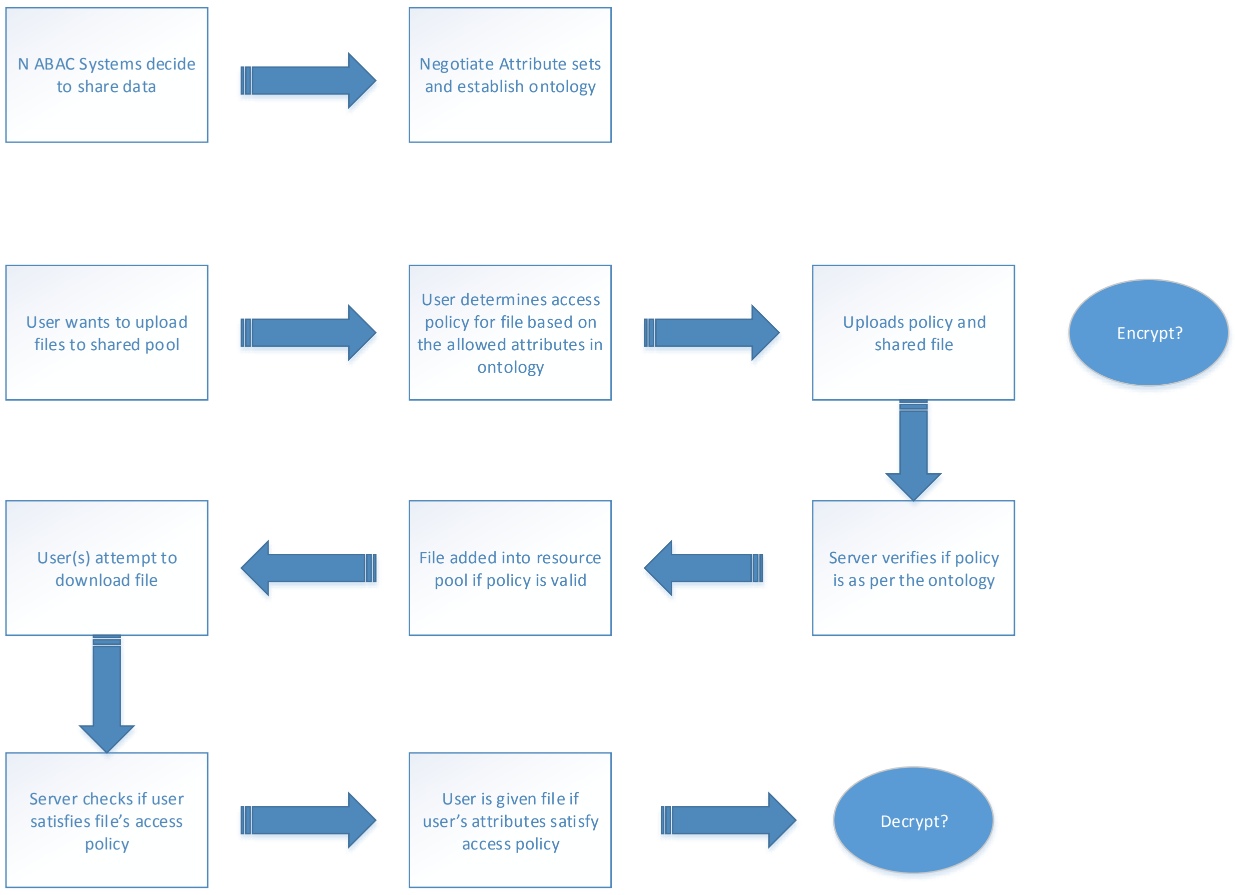

N ABAC Systems decide to share data
Negotiate Attribute sets and establish ontology
User wants to upload files to shared pool
User determines access policy for file based on the allowed attributes in ontology
Uploads policy and shared file
Encrypt?
Server verifies if policy is as per the ontology
File added into resource pool if policy is valid
User(s) attempt to download file
Server checks if user satisfies file's access policy
User is given file if user's attributes satisfy access policy
Decrypt?

Figure 5.1: Workflow

if the access policy (in this case a SPARQL query) is as per the defined ontology and saves it as the access policy for that particular file if it is valid. Once a file is shared, other users have the option of attempting to download the file. When a user tries to download a shared file, the server retrieves the corresponding access policy and executes the query to retrieve the list of users that satisfy the access policy. If the user who is attempting to download the file is a part of this list, then the file is allowed to be downloaded.

5.1 Demo Scenario

The scenario chosen for the testing is as follows.

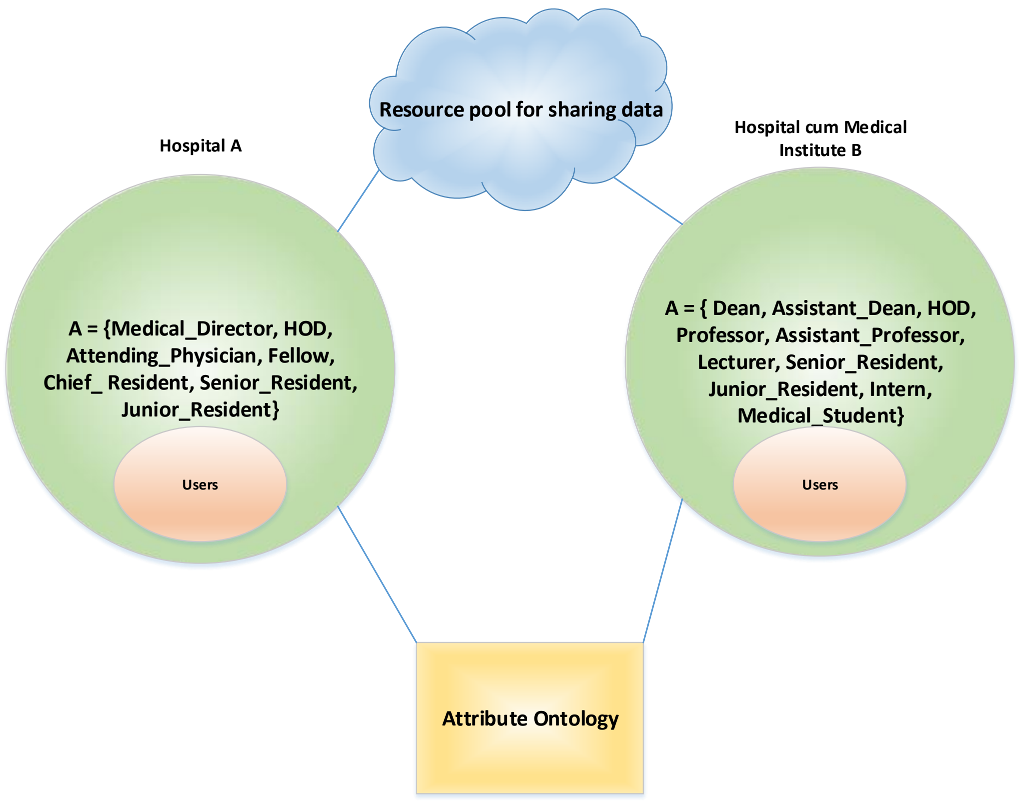

Resource pool for sharing data
Hospital A
Hospital cum Medical Institute B
A = {Medical_Director, HOD, Attending_Physician, Fellow, Chief_ Resident, Senior_Resident, Junior_Resident}
Users
A = { Dean, Assistant_Dean, HOD, Professor, Assistant_Professor, Lecturer, Senior_Resident, Junior_Resident, Intern, Medical_Student}
Users
Attribute Ontology

Figure 5.2: Demonstration

Consider a medical healthcare environment. There are two organisations, a Hospital A that is currently treating a patient, and a Hospital cum Medical Institute B that is a medical college in addition to being a hospital. The attribute sets of the two organisations are depicted in Figure 5.2. As is apparent from the figure, both the organisations have a different manner of categorizing their users and have different sets of literals for describing the possible attributes for a user in their own environment. For example, B has additional attributes that describe the level of the personnel involved in teaching in the institute ($Professor, Assistant\_Professor$, etc). These personnel are also considered Doctors in the system.

Now consider the scenario where a patient who is currently being treated in Hospital A wants to shift the treatment to B for location or personal preference. The medical records of the said patient need to be transferred from A to B. This may seem trivial at first, but legal requirements like HIPAA [18] entail that the data should not be available for public access.The law stipulates that a consent be obtained from the patient as to who should be allowed access to the medical records. Usually the attending doctor or team of doctors are allowed access to the medical records. Medical transcription companies are also allowed access to the data, but not in its entirety. Some places may have additional laws that require further consent to share sensitive data. For example in New York, an additional authorization is needed that explicitly states that the patient wants the facility to release records containing information on HIV diagnosis or treatment. With all these considerations in mind, we need a mechanism to specify access policies for the sensitive data and also keep it flexible.

Given below is the ontology that is established for the scenario depicted in Figure 5.2.

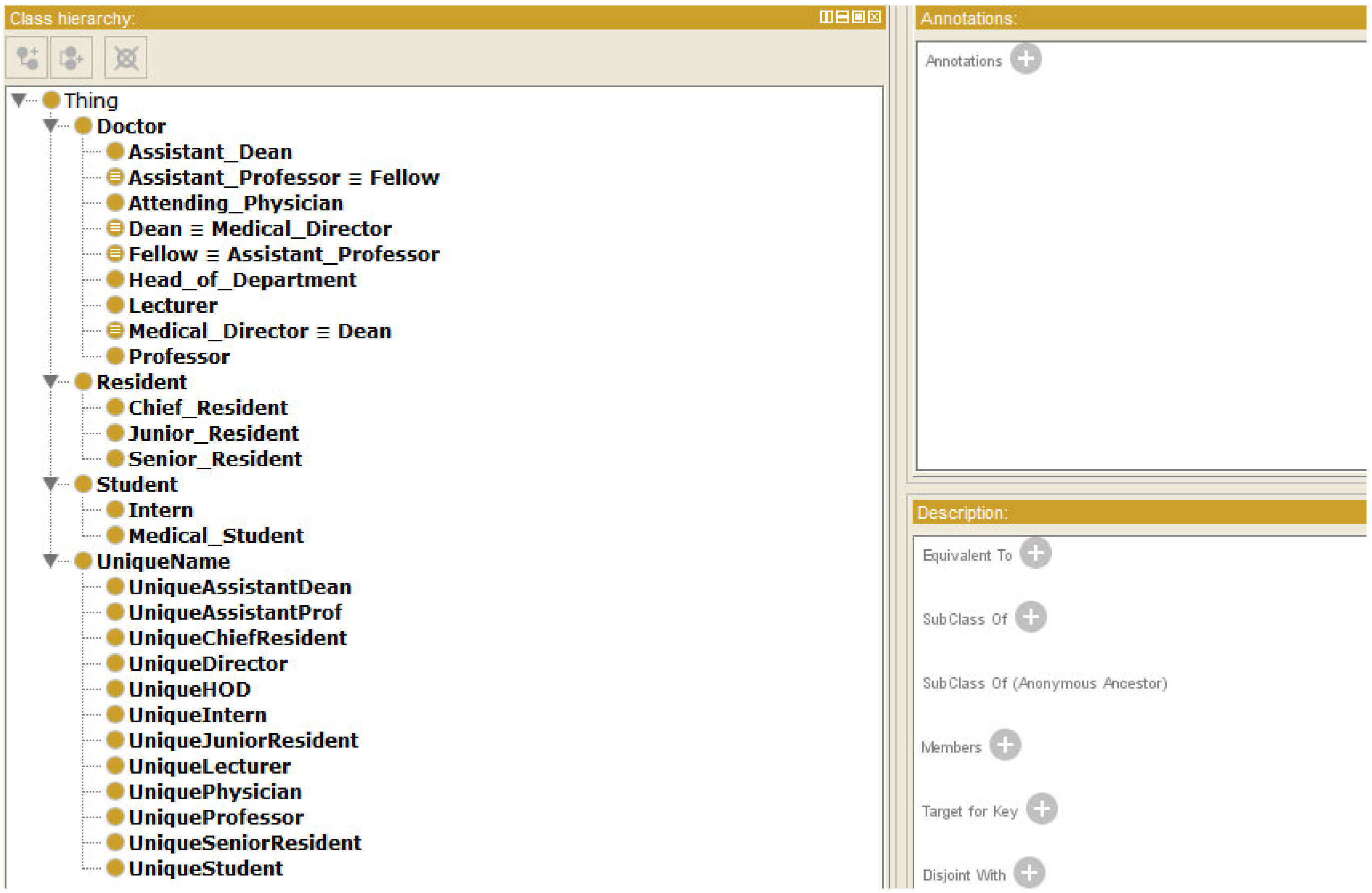
```
<?xml version="1.0"?>


<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY OntologyABAC_Demo "http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#" >
]>

  <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
      OntologyABAC_Demo.owl#Assistant_Dean">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Doctor"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Assistant_Professor">
        <owl:equivalentClass rdf:resource="http://www.semanticweb.org/ontologies
            /2014/2/OntologyABAC_Demo.owl#Fellow"/>
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Doctor"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Attending_Physician">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Doctor"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Chief_Resident">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Resident"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Dean">
        <owl:equivalentClass rdf:resource="http://www.semanticweb.org/ontologies
            /2014/2/OntologyABAC_Demo.owl#Medical_Director"/>
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Doctor"/>
    </owl:Class>
```

```
<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Doctor"/>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Fellow">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Doctor"/>
</owl:Class>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Head_of_Department">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Doctor"/>
</owl:Class>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Intern">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Student"/>
</owl:Class>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Junior_Resident">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Resident"/>
</owl:Class>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Lecturer">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Doctor"/>
</owl:Class>

<owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
    OntologyABAC_Demo.owl#Medical_Director">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Doctor"/>
</owl:Class>
```

```
    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Medical_Student">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Student"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Professor">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Doctor"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Resident"/>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Senior_Resident">
        <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ontologies/2014/2/
            OntologyABAC_Demo.owl#Resident"/>
    </owl:Class>

    <owl:Class rdf:about="http://www.semanticweb.org/ontologies/2014/2/
        OntologyABAC_Demo.owl#Student"/>
```

The attributes have also been classified in order to have more generic specification of access policy. For example, if some data is to be shared with all Doctors, only the attribute *Doctor* can be specified, and all the sub-classes of *Doctor* in the ontology will automatically satisfy the access policy. A more simpler view of the ontology structure in the Protege Ontology Editor [13] shown in Figure 5.3.

Considering that the patient records are accessible to a *Fellow* and the *Chief_Resident* in Hospital A. The patients wants the same authorization to be carried forward to the Institute B. For brevity, I am not considering the specialisation of the doctor and the specific department to which he/she belongs to. A sample access policy in Hospital A is expressed in SPARQL for the ontology described above.



```
Thing
  Doctor
    Assistant_Dean
    Assistant_Professor ≡ Fellow
    Attending_Physician
    Dean ≡ Medical_Director
    Fellow ≡ Assistant_Professor
    Head_of_Department
    Lecturer
    Medical_Director ≡ Dean
    Professor
  Resident
    Chief_Resident
    Junior_Resident
    Senior_Resident
  Student
    Intern
    Medical_Student
  UniqueName
    UniqueAssistantDean
    UniqueAssistantProf
    UniqueChiefResident
    UniqueDirector
    UniqueHOD
    UniqueIntern
    UniqueJuniorResident
    UniqueLecturer
    UniquePhysician
    UniqueProfessor
    UniqueSeniorResident
    UniqueStudent
```

Figure 5.3: Ontology Structure

```
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 ont: <http://www.semanticweb.org/ontologies/2014/2/OntologyABAC_Demo.owl#>
SELECT ?x
        WHERE {
                { ?x rdf:type ont:Fellow }
                UNION
                {?x rdf:type ont:Chief_Resident }
        }
```

Now we use the same access policy without changing it for authorization in Institute B. The ontology reasoner detects that attribute *Fellow* is equivalent to the attribute $Assistant\_Professor$ from the attribute set of Institute B. Therefore any user from Institute B who has the attribute $Assistant\_Professor$ would be able to access the data.

This would work fine with the ABAC system. Now for additional security let us add CPABE encryption to the system. In Figure 5.3, we see a set of classes that represent the unique names of the attributes. All classes that are equivalent to each other are mapped to the same unique name class using the *hasUniqueName* object property. The attribute names in the ABAC policy mentioned above are now replaced with their unique names for the ABE encryption policy. Therefore, the equivalent encryption policy for the above ABAC policy would be

UniqueAssistantProf or UniqueChiefResident

This encryption policy would be valid for both the environments since the unique names are used by the attribute authorities to associate the private keys with attributes. Therefore, as long as the attributes of all the member organizations are mapped correctly to their unique names, we can use ABE techniques for providing additional security in this distributed environment.

Chapter 6

PERFORMANCE EVALUATION

In this section we will evaluate the performance of the ontology-based approach as against not using ontologies in ABAC.

Using the same application scenario described earlier, we will evaluate how using ontologies is more effective. The two medical institutions have their own sets of attributes and they need to share the sensitive data with each other.

6.1 Without Using Ontologies

Hospital A wants to transfer the medical record of a patient to Hospital B. The data has an access policy P_1 over the set of attributes A_1 . If this data is to be shared, then the access policy P_1 has to be translated into an equivalent policy P_2 over the set of attributes A_2. So now the Attribute Authority of Hospital A needs to ask for the attribute set A_2 from the Attribute Authority of Hospital B. Once the set is received, it would need to create a mapping between A_1 and A_2 so that the attributes of A can be mapped to their corresponding attributes in A_2. Once the mapping is created, P_1 is translated into P_2 by replacing the attributes in P_1 with their corresponding attributes from A_2. Therefore there are three operations that need to be performed before any data can be shared between the two hospitals namely *Fetch, Map, Translate.* Considering that the time taken for *Fetch* operation is T_F, time for *Map* operation is T_M and time for *Translate* operation is T_T, then the total time taken for the entire process would be

$$T_{total} = T_F + T_M + T_T$$

In addition to this, if we consider the use of Attribute Based Encryption to encrypt the data, then the encrypted data would need to be decrypted first and then re-encrypted using the attributes from A_2 to specify the encryption policy then the equation becomes

$$T_{total} = T_F + T_M + T_T + 2T_E + T_D$$

where T_E and T_D are the times taken for encryption and decryption respectively. Note that the initial encryption operation is also accounted for since it is a part of sharing the data.

6.2 Using Ontologies for Attribute Management

Initially all the hospitals that would be sharing any form of data would establish a common ontology which would also define the equivalence of the different attributes belonging to the various systems. This process executes only once initially. If a new hospital or medical institution wants to join the share resource pool, then it would add its own attributes in the mapping already created. So now when Hospital A wants to transfer the medical record of a patient to Hospital B, there is no need to perform the *Fetch*, *Map* or *Translate* operations since the ontology already has the mapping created and the attribute equivalence defined in the ontology removes the need for translation of access policies. The encryption using the ontology based approach would use the unique names of the attributes to specify the encryption policies. Since equivalent attributes map to the same unique name, it would remove the need to decrypt and re-encrypt the data again. Therefore only the initial encryption would need to be considered and the total time taken thus becomes

$$T_{total} = T_E$$

The data for the time taken to encrypt and decrypt for different number of attributes was collected and based on that we see the graph below. Note that this graph does not consider the T_F,T_M and T_T values.

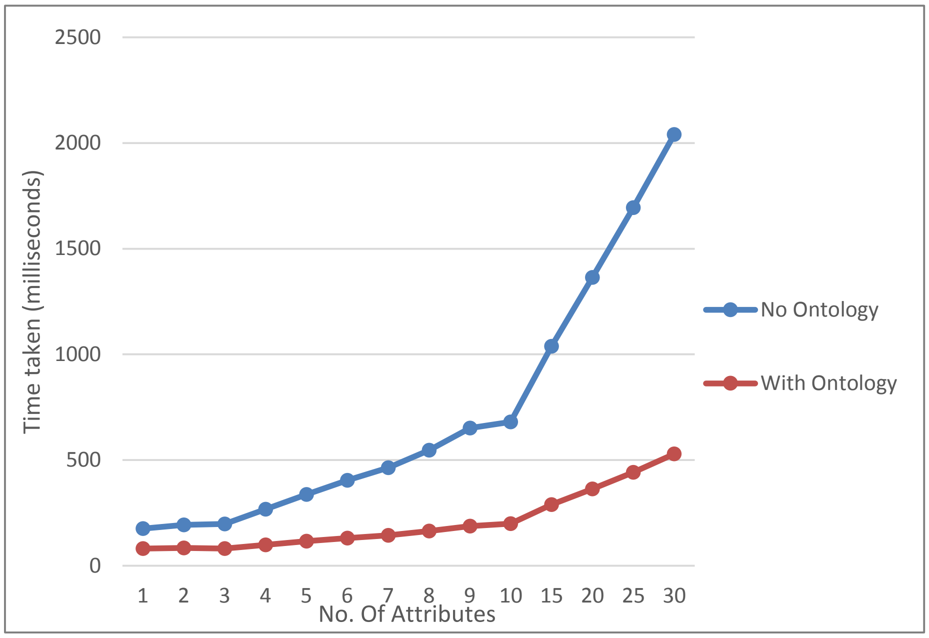


Figure 6.1: Time Consumption for Records Transfer

Comparing the T_{total} values of both the approaches, it is evident that the gain in terms of computation time is significant. This becomes more valuable in a large scale environment when the computing resources are already overloaded and even a minor improvement leads to great performance benefits. Since ABE operations are known to be computationally intensive, the reduction in the number of encryption and decryption operations saves a significant amount of computation power.

In terms of the storage overhead, if we do not use ontologies, then multiple copies of the encrypted data would need to be stored in the resource pool, one for each of the hospital that is one of the intended recipients of the shared data. The encryption policy of these would depend on the target institution and would use the target institution's set of attributes to specify the access policy. When we use ontologies, only a single copy can be stored in the pool since the policy will not change for each of the target institutions. This is beneficial especially when a large number of institutions are a part of the pool and many different types of data is being shared. The reduction in the number of copies allows more data to be shared in the pool without having to swap out existing ones.

Chapter 7

CONCLUSION AND FUTURE WORK

In this thesis we analysed the ABAC model of access control and a new approach to attribute management in ABAC has been proposed. The concept of ontologies is introduced to provide a flexible and scalable mechanism to manage attributes in a distributed manner. With the proposed design, multiple ABAC systems can share data and resources among themselves without needing specific knowledge about the other systems' attribute sets and without compromising the access control enforcement. The design was made even more secure by using CP-ABE to encrypt the data using the access policies specified by the data owners and the ontology design was modified to easily accommodate CP-ABE without needing to encrypt the data again. The validity of the design has been confirmed by creating a demo scenario around it and verifying all the claims made earlier. A web-based implementation was created to test the demo scenario and the claims about the scalability, flexibility and reliability of the system were verified. The major contribution of this research has been a mechanism that makes ABAC much more scalable, thereby making it easy to use in a large scale distributed computing environment (for example a Cloud). The distributed structure of the design allows each system to manage its own attributes independently without affecting the overall data sharing setup.

There is a lot of scope for future work in this area. Some of the areas where future research can focus are

- Modify existing ABE algorithms to allow attribute equivalence so that different user's belonging to different attribute sets can share encrypted data without having to decrypt and encrypt again.

- The efficiency of the system is highly dependent on the structure of the generated ontology. Therefore an optimal structure of the attribute ontology is needed for the system to work efficiently. A methodology is needed that would determine the optimality of the ontology structure, and if possible develop a mechanism that would generate an optimal structure for the ontology that is being negotiated between the ABAC systems.

- Develop mechanism to allow distributed ontologies. Instead of having a single ontology for the whole setup, a mechanism can be developed for allowing the use of multiple ontologies for the attribute structure.

- In the current design, the ontology is assumed to have been setup offline or by an initial negotiation between the different Attribute Authorities. An algorithm for dynamic setup of ontology can be developed where a attribute ranking mechanism can be developed to detect attribute equivalence and create an ontology given n attribute sets.

Bibliography

[1] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, “Guide to attribute based access control (abac) definitions and considerations,” National Institute of Standards and Technology, Tech. Rep., 2014.

[2] R. S. Sandhu, “Role-based access control,” *Advances in computers*, vol. 46, pp. 237–286, 1998.

[3] R. Sandhu, D. Ferraiolo, and R. Kuhn, “The nist model for role-based access control: towards a unified standard,” in *ACM workshop on Role-based access control*, vol. 2000, 2000.

[4] E. Yuan and J. Tong, “Attribute based access control (abac) for web services,” in *Proceedings of the IEEE International Conference on Web Services (ICWS’05)*, 2005.

[5] T. R. Gruber, “A translation approach to portable ontology specifications,” Knowledge Systems Laboratory, Stanford University, Tech. Rep., 1993.

[6] (2014) Rdf schema 1.1. [Online]. Available: http://www.w3.org/TR/rdf-schema/

[7] S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. (2004) Owl web ontology language reference. [Online]. Available: http://www.w3.org/TR/owl-ref/

[8] A. Sahai and B. Waters, “Fuzzy identity-based encryption,” in *Advances in Cryptology–EUROCRYPT 2005.* Springer, 2005, pp. 457–473.

[9] Y. Zhu, D. Ma, C.-J. Hu, and D. Huang, “How to use attribute-based encryption to implement role-based access control in the cloud,” in *Proceedings of the 2013 international workshop on Security in cloud computing.* ACM, 2013, pp. 33–40.

[10] J. Hur and D. K. Noh, “Attribute-based access control with efficient revocation in data outsourcing systems,” *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 7, pp. 1214–1221, 2011.

[11] S. Yu, C. Wang, K. Ren, and W. Lou, “Achieving secure, scalable, and fine-grained data access control in cloud computing,” in *INFOCOM, 2010 Proceedings IEEE.* Ieee, 2010, pp. 1–9.

[12] (2008) Sparql query language for rdf. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/

[13] Protege ontology editor. [Online]. Available: http://protege.stanford.edu/

[14] M. Horridge, H. Knublauch, A. Rector, R. Stevens, and C. Wroe, "A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0," *University of Manchester*, 2004.

[15] Pellet: Owl 2 reasoner for java. [Online]. Available: http://clarkparsia.com/pellet/

[16] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2007.

[17] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

[18] Health information privacy. [Online]. Available: http://www.hhs.gov/ocr/privacy/

APPENDIX A

CIPHERTEXT-POLICY ATTRIBUTE BASED ENCRYPTION

The foundation of ABE is bilinear pairing computation. Assume two groups: an additive group G_0 and a multiplicative group G_1 with a same large prime order p. Discrete Logarithm Problem is difficult in both of them. We define a bilinear map $e : \mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. This map has three properties:

- Bilinearity: $e(P^a, Q^b) = e(P, Q)^{ab}$, for any $P, Q \in \mathbb{G}_0$ and $a, b \in \mathbb{Z}_p$;
- Nondegeneracy: $e(g, g) \neq 1$, where g is the generator of $\mathbb{G}_0$;
- Efficiency: Computing the pairing can be efficiently achieved.

In CP-ABE, there are three types of keys: master key, public key and private key. A TTP is required to generate a set of public parameters and securely store the master key. The TTP will not be involved in the network communication. It can be offline all the time. The scheme of CP-ABE consists of four basic algorithms:

- **Setup**
  The TTP chooses two random exponents $\alpha, \beta \in \mathbb{Z}_p$. A public key is formatted as

  $$PK =< G_0, g, h, f, e(g, g)^\alpha >$$

  The master key is

  $$MK = (\beta, g^\alpha).$$

  Here $h = g^\beta$, $f = g^{\frac{1}{\beta}}$. The public key is published by the TTP before deployment.

- **KeyGen**
  This algorithm is used to generate private keys based on its inputs: the master key (MK) and a set of attributes S. The TTP runs the **KeyGen** algorithm once to generate a private key according to attributes assigned to that node. This private key is then used by the user to decrypt the ciphertext using the Decrypt algorithm. To generate the private key, the algorithm first chooses a random number $r \in Z_p$, and then a random number $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. The secret key is then computed as

  $$SK = (D = g^{(\alpha+r)/\beta}, \forall j \in S : D_j = g^r.H(j)^{r_j}, D'_j = g^{r_j})$$

- **Encrypt**
  The inputs of this algorithm are the public key, the message M and a policy tree T over the set of allowed attributes. The algorithm encrypts M and outputs ciphertext CT such that only those users whose private keys have attributes that satisfy the policy represented by T will be able to decrypt it.

  1. The degree of the secret polynomial p_x is set to be $d_x = k_x - 1$

2. For root node r, choose a random value s in $\mathbb{Z}_p$. Set $p_r(0) = s$. An element d_r in $\mathbb{Z}_p$ is chosen randomly to completely define p_r. For any other non-leaf node x, set its secret to be one secret share of its parent node, i.e. $p_x(0) = p_{parent(x)}(index(x))$, and randomly choose element d_x to completely define p_x.
3. Define a hash function $H : \{0,1\}^* \rightarrow G$ which is modelled as a random oracle. For each leaf node x, assign the value

   $$E_x = g^{p_x(0)}, E'_x = H(att(x))^{p_x(0)}$$

   The ciphertext is then constructed using the access structure T

   $$CT = (T, \widetilde{C} = Me(g,g)^{\alpha s}, C = h^s, \forall y \in Y : C_y = g^{q_y(0)}, C'_y = H(att(y))^{q_y(0)})$$

- **Decrypt**
  This algorithm takes the ciphertext (which contains the access policy), the public parameters and a private key as inputs. If the attributes in the private key satisfy the access policy of the ciphertext, the algorithm will decrypt the ciphertext and output the original message.

  A recursive algorithm DecryptNode(CT, SK, x) is defined that takes as input the ciphertext $CT = (T, \widetilde{C}, C, \forall y \in Y : C_y, C'_y)$, a secret key SK which is associated with a set S of attributes, and a node x from T. If the node is a leaf node, then we take $i = att(x)$. If $i \in S$, then

  $$\begin{aligned} DecryptNode(CT, SK, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= \frac{e(g^r . H(i)^{r_i}, h^{q_x(0)}}{e(g^{r_i}, H(i)^{q_x(0)}} \\ &= e(g,g)^{rq_x(0)} \end{aligned} \quad \text{(A.1)}$$

  If $i \notin S$, then $DecryptNode(CT, SK, x) = \perp$.

  When x is a non-leaf node, for all the nodes z that are children of x, it calls $DecryptNode(CT, SK, z)$ and stores the output as F_z. Let S_x be an arbitrary k_x-sized set of child nodes z such that $F_z \neq \perp$. If no such set exists, then the function returns $\perp$. Otherwise we compute

$$
\begin{aligned}
F_x &= \prod_{z \in S_x} F_z^{\Delta_{i,S'_x}(0)}, i = index(z), S'_x = index(z) : z \in S_x \\
&= \prod_{z \in S_x} (e(g,g)^{r.q_z(0)})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} (e(g,g)^{r.q_{parent(z)}(index(z))})^{\Delta_{i,S'_x}(0)} \\
&= \prod_{z \in S_x} e(g,g)^{r.q_x(i).\Delta_{i,S'_x}(0)} \\
&= e(g,g)^{r.q_x(0)}
\end{aligned}
\tag{A.2}
$$

The Decrypt algorithm begins by calling the DecryptNode function on the root node R of the tree T. If the tree is satisfied by S we set $A = DecryptNode(CT, SK, r) = e(g,g)^{rq_R(0)} = e(g,g)^{rs}$. It now decrypts by computing

$$\widetilde{C}/(e(C,D)/A) = \widetilde{C}/(e(h^s, g^{(\alpha+r)/\beta})/e(g,g)^{rs}) = M$$