Distributed Policy Decision Points for Electronic Health Records

SARAVANA KUMAR SANKARAMOORTHY

Master's Thesis
Supervisor: Dr. Jim Dowling, SICS
Examiner: Prof. Seif Haridi, KTH/SICS
Stockholm, 12 December 2011
Abstract

The advancement in technology mandates the extensive use of computerized healthcare devices making Electronic Health Records (EHRs) the way to store the patient details. The EHR systems have high availability and security requirements for the storage database. DIGHT is a distributed key-value store architecture being developed at SICS addressing the problems of high availability and scalability, data integrity and confidentiality, accountability, EHR versioning and extensibility.

This Master thesis addresses the authorization requirements of the EHR systems. eXtensible Access Control Markup Language (XACML) is a OASIS standard for general purpose access control policy language designed for managing the access for resources. All of the available open source implementation of Policy Decision Point (PDP) conforms to XACML version 2.0 and retrieves the policies from the traditional file systems.

Sun open source implementation of PDP conforming to XACML 2.0 was evaluated. It was upgraded to conform with XACML 3.0 standards. The XACML Admin Profile for delegation was also implemented. The testing was carried out with a prototype application which accepts text sms from registered doctors through an sms gateway. The application was designed for adding new patient record, medical record to an existing patient and retrieving existing patient and medical records. The application generates the XACML Request and send it to the PDP for evaluation. The XACML policies for authorizing the doctors were stored in MySQL database clusters. The PDP evaluates the request and send the XACML Response back to the application. The application processes the response and send appropriate reply to the sender.

Performance evaluation was carried out with policies stored in database clusters. The thesis also discusses about the possible future enhancements like implementing XACML profile for SAML assertions, implementing the Policy Information Point to fetch attributes.
Dedicated to
my parents Sankaramoorthy and Dhalalakshmi
and
all of my friends
Acknowledgements

First of all, I would like to thank Swedish Institute of Computer Science (SICS) for providing me an opportunity to work on this research project. I would like to express my gratitude to my supervisor, Dr. Jim Dowling for his continual support and guidance throughout this thesis. He has always been around with his constant encouragement and help whenever required.

I am very grateful to my examiner, Professor Seif Haridi for his assistance, guidance and feedback throughout this project. I also owe thanks to Mr. Frej Drejhammar at SICS for his support in this project.

I also wish to thank all of my friends who are my driving force not only for this project but to my entire life.

And finally, I would like to thank my parents and brother Sasikumar without whose encouragement, support and trust, I would not have come this far.

Stockholm, Sweden
12 December, 2011

Saravanakumar Sankaramoorthy
# Contents

1 Introduction ........................................... 1  
   1.1 Problem Statement .................................. 1  
   1.2 Approach ............................................ 2  
   1.3 Report Structure .................................... 2  

2 Background/Related Work ................................. 3  
   2.1 XACML Specification ................................. 3  
      2.1.1 XACML architecture ............................. 3  
      2.1.2 Advantages of XACML ......................... 5  
      2.1.3 XACML Context ................................. 5  
      2.1.4 XACML Policy language model .................. 6  
   2.2 DIGHT Project ...................................... 14  
      2.2.1 DIGHT Overview ................................. 14  
      2.2.2 DIGHT Architecture ............................. 15  
      2.2.3 Cloud distribution architecture ............... 16  
      2.2.4 Implementation Language ...................... 17  

3 Model/Implementation .................................. 19  
   3.1 Enhancements and New features of XACML 3.0 ............ 19  
   3.2 Enhancements Done ................................... 20  
      3.2.1 Policy Administration Point .................... 21  
      3.2.2 Policy Decision Point ........................... 21  
      3.2.3 Policy Enforcement Point prototype .......... 22  

4 Evaluation ............................................. 33  
   4.1 Prototype Evaluation ............................... 33  
      4.1.1 Environment Setup ............................... 33  
      4.1.2 Add Patient Details .............................. 34  
      4.1.3 Add Medical Record .............................. 35  
      4.1.4 Get Patient Details .............................. 36  
      4.1.5 Get Medical Record .............................. 38  
   4.2 Performance Evaluation .............................. 40  
      4.2.1 DIGHT database Setup ........................... 40
5 Conclusion and Future Work
5.1 Conclusion ........................................ 43
5.2 Future Work ....................................... 43

A Samples ............................................. 45
A.1 Policy Used ....................................... 45
A.2 Sample Request ................................. 49
A.3 Sample Response ............................... 50

Bibliography .......................................... 51
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>XACML Architecture</td>
<td>4</td>
</tr>
<tr>
<td>2.2</td>
<td>XACML Context</td>
<td>5</td>
</tr>
<tr>
<td>2.3</td>
<td>XACML Language Model</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>DIGHT Architecture</td>
<td>16</td>
</tr>
<tr>
<td>2.5</td>
<td>DIGHT Cloud Distribution Architecture</td>
<td>17</td>
</tr>
<tr>
<td>3.1</td>
<td>DIGHT sms prototype implementation architecture</td>
<td>27</td>
</tr>
<tr>
<td>3.2</td>
<td>PEP prototype design</td>
<td>28</td>
</tr>
<tr>
<td>3.3</td>
<td>Sequence diagram for adding patient details</td>
<td>29</td>
</tr>
<tr>
<td>3.4</td>
<td>Sequence diagram for adding medical record</td>
<td>30</td>
</tr>
<tr>
<td>3.5</td>
<td>Sequence diagram for retrieving Patient Details</td>
<td>31</td>
</tr>
<tr>
<td>3.6</td>
<td>Sequence diagram for retrieving a medical record</td>
<td>32</td>
</tr>
</tbody>
</table>
## List of Tables

2.1 Target match table ........................................... 11
2.2 AnyOf match table ........................................... 11
2.3 AllOf match table ........................................... 11
2.4 Rule truth table ............................................... 12
2.5 Policy truth table ........................................... 12
2.6 PolicySet truth table ....................................... 13

4.1 Rule truth table ............................................... 34
4.2 Node Address Map ........................................... 40
4.3 Node configuration in cloud ................................ 40
4.4 Key Range-Replication Group ............................... 41
4.5 PEP and PDP configuration .................................. 41
4.6 Performance Values ........................................... 41
Chapter 1

Introduction

The advancement of technology mandates the extensive use of computerized healthcare devices. This leads to the digital storage of data in the form of Electronic Health Records (EHRs) [4]. Most of the EHR systems store the records in the centralized architecture which causes unavailability if the server is crashed. On 11th and 12th of December 2008, Sweden faced such crisis when its 'TakeCare' EHR system for Stockholm and Gotland crashed due to a suspected disk failure [6].

EHR Systems have high availability and security requirements. DIstributed Global Health Technology (DIGHT) [19] is a project being developed at Swedish Institute of Computer Science (SICS) [20], which supports high availability and scalability, data integrity and confidentiality, accountability, EHR versioning and extensibility. DIGHT is a open source distributed key-value store architecture designed for EHR where the patient record is always available for reading and writing and always be secured from unauthorized reading or writing or theft. The authorization safety is achieved using XACML [12]

eXtensible Access Control Markup Language (XACML) is a general purpose access control policy language. It is an OASIS [11] standard which defines syntax in XML [3] to manage access control for resources. It defines the policy language and an access control decision request/response language. The former describes the general access control mechanisms and extension points for defining new functions, data types and combining logic whereas the latter defines how to form the request to ask the permission for a particular resource, how to form the response for the sent request and how to interpret the response [7].

1.1 Problem Statement

SICS is working on DIGHT, a project on building key-value store architecture for storing electronic health records of billion citizens of India. DIGHT needs an authorization mechanism to allow the users to access their specific resource alone. Most of the existing authorization mechanism are proprietary. The authorization
policies defined for one implementation may not suit another implementation. The authorization systems are also centralized and non-standard ones. So there is a risk of unavailability problem. Also the policy written for one implementation may not apply for other implementations unless a standard is followed. There is also a problem in sharing the policies across different locations.

1.2 Approach

This thesis involves developing authorization infrastructure for the patients and health workers to securely access the health records using XACML standards. The thesis develops a distributed Policy Decision Points to authorize the patients and health workers to securely access the health records based on the XACML policies stored in relational database. The Sun implementation of XACML 2.0 Standards [10] is improved to comply with XACML 3.0 standards [17]. The distributed PDP is implemented on top of Kompics [18], a message passing component model developed in SICS. MySQL clusters is used to store the XACML policies and hence replicating them across locations can be made simple.

1.3 Report Structure

The layout of the rest of this report is summarized as follows. The next chapter talks about the background study performed followed by a chapter on the design and implementation details. The consecutive chapter discusses the evaluation of results. The last chapter provides the conclusion of this thesis and possible future works based on this thesis.
Chapter 2

Background/Related Work

This chapter provides the background details necessary to understand this thesis. The thesis requires the background knowledge about XACML standards, DIGHT architecture and Kompics component model.

2.1 XACML Specification

eXtensible Access Control Markup Language (XACML) is an OASIS standard that defines a general purpose policy language and an access control decision request/response language. Most of the existing systems implement access control and authorization in proprietary manner. Hence the programmers and IT technicians configuring these application systems define the access control rather than the business managers. XACML standardizes the method to manage access to the resources across applications using the policy language and request/response language (both written in XML). The policy language has general access control requirements and provides necessary extension points to define new data types, functions and combining logic, etc. The request/response protocol language provides the method to form the query to confirm whether the particular action should be permitted or not, and method to interpret the response. The response should always contain an answer whether the request should be allowed using one of the four values viz Permit, Deny, Indeterminate (when an error occurs or some required attribute is missing) and Not applicable (the particular request cannot be processed).

2.1.1 XACML architecture

The XACML architecture is shown in Fig: 2.1. The major components of the architecture are explained below.

Policy Enforcement Point (PEP)  Policy Enforcement Point (PEP) is the entry point for the authorization system. All the requests to the secured resources
reach the PEP first. The PEP will form the access request based on the user’s attributes, the resource in question, the action requested and other information related to the request. The access request is then sent to the Policy Decision Point (PDP) for making the decision. The decision response is then sent back to the PEP. The PEP is responsible to enforce the authorization decision and obligations returned by PDP. An obligation is an action that should be enforced by the PEP along with the authorization decision. The XACML standard does not specify how to transfer access request from the PEP to PDP. The PEP and PDP can co-exist in the same application system or it may locate many servers away.

**Policy Decision Point (PDP)** Policy Decision Point (PDP) is the important component of the authorization system. The PDP receives the XACML request from the PEP and arrives at the decision considering the available policies and rules within them. The PDP does not evaluate all the policies available. The policies are chosen and evaluated based on the policy target. The policy target contains attributes to be matched by the xacml request attributes. The policy is chosen only if the particular request’s attributes match the target attributes. The detailed procedure of policy evaluation is discussed later in this chapter.

**Policy Administration Point (PAP)** Policy Administration Point (PAP) is the place where policies are written and stored. PDP uses the PAP to get access to the available policies. It also provides restriction in order to prevent unauthorized
access to the policies. Also, it conducts the regular check to maintain the uniqueness of the policy identifiers.

**Policy Information Point (PIP)** Policy Information Point (PIP) is the component where the attribute values are stored and makes it available to PDP. Attribute values are the data that define the characteristics of the attributes of the requester, resources and the actions to be performed on the resources. Resource may refer to any document, information, file or any data.

### 2.1.2 Advantages of XACML

- XACML is a standard language. It has been reviewed by a wide community of experts and users reducing the complexity involved in designing a new language. Once XACML is widely deployed it will become easier to interoperate with other applications using the same standard language.

- It is generic. Hence policy designed for one environment can be ported to any other environment without any fuss. This makes policy management much easier.

- It is distributed. This allows a policy to refer the policies stored in different locations written and maintained by different author groups. XACML facilitates intelligent combining algorithms to combine those policies and arrive at a single decision.

- It is powerful. XACML standard itself provides wide variety of data types, functions and algorithms to combine various rules and policies. There are also many communities working on extensions and profiles to hook XACML into other standards like SAML and LDAP.

### 2.1.3 XACML Context

![Figure 2.2. XACML Context](image)
XACML is designed with intention to cover variety of application environments. The XACML specification covers the area shown in grey colour in the Fig: 2.2 insulating it from the application environment. XML Schema is used to define XACML context which describe the canonical representation of the inputs and outputs of the PDP. XACML policy attributes’ references may be in the form of XPath expressions over the <Content> elements of the context or by attribute designators identified by the Category, datatype, attribute identifier and optionally the issuer of the attribute. The XACML implementation should be responsible for conversion of attribute representations in the application environment and the XACML context.

2.1.4 XACML Policy language model

The language model used in XACML is shown in Fig: 2.3

The main elements of the model are

- Rule
- Policy
- PolicySet

Rule

Rule is the most elementary unit of the XACML policy. It cannot be represented in isolation. It can exist only within a Policy element. The Rule is evaluated based on its contents. Rule is uniquely identified by RuleId attribute represented as URI, URN or URL. PAP is responsible to ascertain that no two rules visible to the PDP has the same RuleId. Possible contents of a Rule are

1. zero or one Target
2. zero or one Condition
3. an Effect
4. zero or more Obligation Expressions
5. zero or more Advice Expressions

Rule Target The target decides the set of requests to which the Rule will apply, through a set of predicates which should be matched by the attributes passed in the XACML Request context. If the Target is missing, the enclosing Policy’s Target will act as the Rule’s Target. The PDP has to verify whether the matches specified in the Target are matched by the attributes given in the request.
Effect  The Effect of the Rule specifies the intended consequence if the Rule evaluates to True. The possible values allowed are Permit and Deny.

Condition  Condition is the logical boolean expression which refines the applicability of the rule beyond the conditions specified by the target. Hence it is not mandatory to be present.

ObligationExpressions  A rule can contain zero or more Obligation Expressions which will be evaluated into Obligations in the Response context if the rule is evaluated to the same decision as that of the FulfillOn attribute of the Obligation
Expression. The Obligation returned by the PDP should be satisfied by the PEP. If the PEP cannot process the obligation or does not understand it, the access to the resource should be denied. If any of the attribute assignment expression in the Obligation evaluates to Indeterminate, then the resulting policyset, policy or rule is evaluated to Indeterminate. If the evaluated result of Rule does not match the FulfillOn attribute of Obligation Expression, then the Indeterminate decision does not have any effect on the rule, policy or policyset.

AdviceExpressions A rule can contain zero or more Advice Expressions which will be evaluated into Advices in the Response Context if the rule is evaluated to the same decision as that of the AppliesTo attribute of the Advice Expression. Unlike the Obligations, the Advices returned by PDP can be safely discarded by the PEP. Processing the Advice is solely under the discretion of PEP. Advice Expression evaluation is similar to that of evaluation of Obligation Expression.

The PDP evaluation can be viewed as a tree of policysets, policies and rules. Each node of the tree will be evaluated to Permit or Deny. The Obligations and Advices returned by the PDP to PEP includes only the obligations and advices associated with the path where the effect of the evaluation path is the same as that of the effect returned by the PDP.

Policy

Policy is the simplest unit which can be exchanged among the system entities. Policy is uniquely identified by the PolicyId attribute represented as URI, URL or URN. PAP is responsible to ascertain that no two policy visible to the PDP has the same PolicyId. Allowed elements of Policy are

1. a Target
2. one or more Rule
3. a Rule Combining Algorithm identifier
4. zero or more Rule Combiner parameters
5. zero or more Obligation Expressions
6. zero or more Advice Expressions
7. an optional Policy Issuer

The Rule element was described in the above section. The Obligation and Advice Expressions are similar to that of Rule element. The optional Policy Issuer specifies the attributes of the issuer of the policy. The remaining elements are described in the following subsections.
Policy Target  The PolicySet, Policy and Rule element contains a Target element that specifies a set of requests to which it applies. Two logical methods can be applied to form the Target elements. One is forming the outer Target element as the union of the Target elements of the referenced policies, policysets and rules (i.e., inner components). Other method is forming the outer Target element as the intersection of the Target elements of the referenced policies, policysets and rules.

Rule Combining Algorithm  Rule-combining algorithm specifies the procedure to combine the evaluation results of the component rule elements of the policy. The decision of the PDP placed in the response context is defined by the rule combining algorithm of the policy. The policy can contain combiner parameters which are passed as arguments to rule combining algorithm.

PolicySet  
PolicySet is the top level element of the XACML policy schema. PolicySet is uniquely identified by the PolicySetId attribute represented as URI, URL or URN. PAP is responsible to ascertain that no two policysets visible to the PDP has the same PolicySetId. Allowed elements of a PolicySet are

1. a Target
2. zero or more Policy
3. zero or more PolicySet
4. a Policy Combining Algorithm identifier
5. zero or more Obligation Expressions
6. zero or more Advice Expressions
7. zero or more Policy Combiner parameters
8. an optional Policy Issuer

The Target, Policy, Policy Issuer, Obligation Expressions and Advice Expressions are same as described in the previous sections. PolicySet can enclose other policies and policysets directly or through references. The Policy and PolicySet references are identified by the PolicyId and PolicySetId respectively.

Policy Combining Algorithm  Policy-combining algorithm specifies the procedure to combine the evaluation results of the component policy elements of the policyset. The decision of the PDP placed in the response context is defined by the policy combining algorithm of the policyset. The policyset can contain combiner parameters which are passed as arguments to policy combining algorithm. The policysets enclosed within a policyset is treated as similar to policy by the policy combining algorithm.
Attributes

XACML processing is performed with attributes. All the attributes have a unique identifier with values of known data types. Attributes can have optional issuer value. Attributes mainly characterizes the subject, action, resource or environment on which the access request is made. The file, name of the accessor, current time, the action to do on a particular file are examples of attribute values. When the request is sent from PEP to PDP, the request is formed with the attributes. The attributes sent in the request are compared with the attribute values present in the policy to make the access decisions.

The attributes in the request or from some other sources can be retrieved using two mechanisms, AttributeDesignator and AttributeSelector.

AttributeDesignator  An AttributeDesignator allows a policy to denote an attribute using its identifier, category, datatype and optionally the issuer. The PDP will then look for the particular value in the request or elsewhere like database if the value is not found in the request.

AttributeSelector  An AttributeSelector allows a policy to resolve the attribute using XPath query. A datatype, XPath expression and the category to which the attribute belong to are provided to the AttributeSelector and these can be used to resolve the attribute values from the request document or any other document.

Both the AttributeDesignator and AttributeSelector can return multiple values since multiple value matches can be found for an attribute in the request or elsewhere. Hence XACML provides a special kind of attribute called Bag attribute. Both AttributeDesignator and AttributeSelector always return bag attribute even if there is only one match for the attribute. In the case that no match found for an attribute, an empty bag is returned. AttributeDesignator and AttributeSelector use a flag(MustBePresent attribute) to raise an error if no match is found for an attribute. Bag attribute is an unordered collection of basic datatype elements and can contain duplicate elements.

Function

Once the Bag value is obtained, PDP should somehow compare each values of the bag with the expected value to make the access decision. XACML provides a set of powerful functions to handle any combination of attribute values and can return any attribute value available in the XACML system. Functions can be nested, that is a function can operate on the output of the other functions. Custom functions can be defined to enhance the XACML language.

Most of the functions are defined to work with single element of a particular datatype. But the AttributeDesignator and AttributeSelector always return a bag of values. XACML provides a set of functions to handle bags. [type]-one-and-only function is available for each datatype specified in XACML which checks whether
the bag contains only one element of the specific type. If zero or multiple values are present, these set of functions will return an error. But these functions are not needed in targets since PDP is designed such that each element of the bag is compared with the attribute value using the specified match function.

**Functional Requirements**

**Target Evaluation**  An empty Target matches all the requests. If the target is not empty, then all the AnyOf elements in the target should match the attribute values in the request context to evaluate to **Match**. If any one of the AnyOf element does not match, then the target evaluates to **No Match**. Otherwise, the target evaluates to **Indeterminate**. The target match table is shown below in Table: 2.1.

<table>
<thead>
<tr>
<th>AnyOf value</th>
<th>Target value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All &quot;Match&quot;</td>
<td>&quot;Match&quot;</td>
</tr>
<tr>
<td>Atleast one &quot;No Match&quot;</td>
<td>&quot;No Match&quot;</td>
</tr>
<tr>
<td>Otherwise</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

The AnyOf shall match values in the request context if at least one of the AllOf elements evaluates to Match. AnyOf match table is shown below in Table: 2.2

<table>
<thead>
<tr>
<th>AllOf value</th>
<th>AnyOf value</th>
</tr>
</thead>
<tbody>
<tr>
<td>At least one &quot;Match&quot;</td>
<td>&quot;Match&quot;</td>
</tr>
<tr>
<td>None matches and at least one &quot;Indeterminate&quot;</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
<tr>
<td>All &quot;No Match&quot;</td>
<td>&quot;No Match&quot;</td>
</tr>
</tbody>
</table>

The AllOf shall match values in the request context if all the Match elements evaluates to True. AllOf match table is shown below in Table: 2.3

<table>
<thead>
<tr>
<th>Match value</th>
<th>AllOf value</th>
</tr>
</thead>
<tbody>
<tr>
<td>All &quot;True&quot;</td>
<td>&quot;Match&quot;</td>
</tr>
<tr>
<td>No False and at least one &quot;Indeterminate&quot;</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
<tr>
<td>At lease one &quot;False&quot;</td>
<td>&quot;No Match&quot;</td>
</tr>
</tbody>
</table>
**Rule Evaluation** A Rule has a value which is calculated by evaluating its contents. Rule evaluation is the combined evaluation of Rule’s target and the optional condition. The rule truth table is shown below in Table: 2.4

<table>
<thead>
<tr>
<th>Target value</th>
<th>Condition value</th>
<th>Rule value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot; or No Target</td>
<td>&quot;True&quot;</td>
<td>&quot;Match&quot; or No Target</td>
</tr>
<tr>
<td>&quot;Match&quot; or No Target</td>
<td>&quot;False&quot;</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Match&quot; or No Target</td>
<td>&quot;Indeterminate&quot;</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
<tr>
<td>&quot;No Match&quot;</td>
<td>Don’t care</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>Don’t care</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

If the Target evaluates to 'No Match' or 'Indeterminate", then the rule evaluates to "NotApplicable" and "Indeterminate" respectively irrespective of the condition evaluation. In the above cases, the condition is not evaluated. If the Target evaluates to "Match" or there is no Target and the condition evaluates to "True", then rule evaluates to the Rule’s Effect attribute value.

**Policy Evaluation** The value of a policy is evaluated by its contents. A policy’s value is evaluated by evaluating its target and member rules. The policy’s applicability is evaluated by the applicability of its target. If the target is matched, then the rules are evaluated and the policy’s value is determined by the mentioned rule-combining algorithm. If the Target evaluates to "No Match", then the policy’s value is evaluated as "NotApplicable" without considering evaluation of its rules. If the Target evaluates to "Indeterminate", then the policy’s value is also "Indeterminate". The Policy truth table is shown below in Table: 2.5

<table>
<thead>
<tr>
<th>Target value</th>
<th>Rule value</th>
<th>Policy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one rule value is its Effect</td>
<td>determined by the rule-combining algorithm</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>All rules evaluate to &quot;NotApplicable&quot;</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one rule value is &quot;Indeterminate&quot;</td>
<td>determined by the rule-combining algorithm</td>
</tr>
<tr>
<td>&quot;No Match&quot;</td>
<td>Don’t care</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>Don’t care</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

A rule value of 'At least one rule value is its Effect' denotes either there are no rules in the policy, or atleast one of the contained rules in the policy evaluates to
its Effect (refer 2.4). A rule value of "All rule values evaluate to NotApplicable" is applied if all the contained rules evaluate to "NotApplicable". If none of the contained rules in the policy is applicable to the request and one or more rule returns a value of "Indeterminate", then the rule value will be "At least one rule value is Indeterminate".

If the target value is "Match" and the rule value is "At least one rule value is its Effect" or "At least one rule value is Indeterminate", then the rule-combining algorithm determines the policy value. Rule-combining algorithms defined by XACML 3.0 does not take parameters. But user defined combining algorithms may be implemented taking parameters. Hence the parameter values associated with the rules has to be considered when evaluating the policy. The parameters and their types should be defined in the specification of the combining algorithm.

**PolicySet Evaluation** The value of a policyset is evaluated by its contents. A policyset value is evaluated by evaluating its target and member policy and policiesets. The policyset’s applicability is evaluated by the applicability of its target. If the target is matched, then the contained policy and policiesets are evaluated and the policyset’s value is determined by the mentioned policy-combining algorithm. If the Target evaluates to "No Match", then the policyset’s value is evaluated as 'NotApplicable' without considering evaluation of its contained policies and policiesets. If the Target evaluates to "Indeterminate", then the policyset’s value is also 'Indeterminate'. The PolicySet truth table is shown below in Table: 2.6

<table>
<thead>
<tr>
<th>Target value</th>
<th>Policy value</th>
<th>PolicySet value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one policy value is its <strong>Decision</strong></td>
<td>determined by the policy-combining algorithm</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>All policies evaluate to 'NotApplicable'</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Match&quot;</td>
<td>At least one policy value is 'Indeterminate'</td>
<td>determined by the policy-combining algorithm</td>
</tr>
<tr>
<td>&quot;No Match&quot;</td>
<td>Don’t care</td>
<td>&quot;NotApplicable&quot;</td>
</tr>
<tr>
<td>&quot;Indeterminate&quot;</td>
<td>Don’t care</td>
<td>&quot;Indeterminate&quot;</td>
</tr>
</tbody>
</table>

A policy value of "At least one policy value is its Decision" denotes either there are no policy or policiesets within the policyset, or atleast one of the contained policies or policiesets in the policyset evaluates to its Decision (refer Table: 2.5). A policy value of "All policies evaluate to NotApplicable" is applied if all the contained policies and policiesets evaluate to NotApplicable. If none of the contained policies and policiesets in the policyset is applicable to the request and one or more policy returns a value of "Indeterminate", then the policyset value will be "At least one policy value is Indeterminate".
If the target value is "Match" and the policy value is "At least one policy value is its Decision" or "At least one policy value is Indeterminate", then the policy-combining algorithm determines the policyset value. Policy-combining algorithms defined by XACML 3.0 does not take parameters. But user defined combining algorithms may be implemented taking parameters. Hence the parameter values associated with the policy has to be considered when evaluating the policyset. The parameters and their types should be defined in the specification of the combining algorithm.

2.2 DIGHT Project

DIGHT is a distributed database architecture to store EHR data. It uses the cloud architecture which synchronizes the data from the local database of the hospitals. The DIGHT architecture is designed such that EHR data stored in the legacy database is synchronized safely, securely and stored in the cloud so that the other hospitals in the network can securely access the data. EHR data must always be available even in the case of node crash or partial network failure. Since all the hospitals have to access the EHR data, DIGHT is designed to map the legacy data from the database to standardized EHR data like HL7 or DICOM.

The distributed database in the DIGHT cloud is formed with a set of interconnected group of nodes. Each group of nodes represents a database instance. Since there are more than one node in a group, the data access is available even in the case of partial network failure. This is achieved by replicating the database contents across multiple nodes.

2.2.1 DIGHT Overview

Distributed database of DIGHT architecture provides the high availability, extensibility, scalability, integrity and EHR versioning. Hence, the data is always available for reading and writing. EHR versioning avoids the illegal reading, writing or access of the data.

Data Integrity and Confidentiality

Data confidentiality provides security against unauthorized access to information and Data Integrity provides security against improper alterations to information. Though existing databases used to store EHRs provide authentication, authorization and encryption mechanisms, the following additional threats are possible to violate the data confidentiality and data integrity of the EHR information:

- unencrypted data access from the physical media or disk
- snooping unencrypted data over the network
- breaking encryption on the encrypted data
2.2. DIGHT PROJECT

- identity spoofing
- circumventing authorization and authentication mechanism

Accountability

All the access and change to the EHR information should be monitored and recorded ie all the information must be non-repudiable. Audit log is created and maintained for all the EHR operations to achieve this. DIGHT provides the database support to create the audit trails easily and maintain the versioning of database objects.

Dependability

Dependability is defined by the properties of high availability, reliability and maintainability. In DIGHT, dependability is achieved by replicating the information on multiple nodes which are geographically distributed and run on different networks. Since there is no single point of failure and the nodes run on different networks, reliability and maintainability properties are also provided.

Scalability

DIGHT should be capable of handling both millions of patients and millions of transactions per day. To enable scalability in DIGHT, EHRs are partitioned over a federated architecture, enabling parallel read and write EHR operations at many distributed sites.

2.2.2 DIGHT Architecture

The DIGHT architecture shown in Fig: 2.4 is built with many layers. Each layer handles the responsibility of processing some part of a EHR request. EHR API is the top layer responsible for handling set of standard operations like reading, writing and updating EHRs. EHR management system and EHR processing systems are written using this layer. EHR API can be written in any of the known standards including HL-7 and Sweden’s National Infrastructure process.

A plugin which can be Process Model or Information Model, processes the EHR API requests and converts them to Object API requests. The requests on the EHR API should support the security authenticator model based on certificates. Object API provides methods to read and write data in the database. Each hospital has a server responsible for handling Object API operations. If the operations received at the server has to be applied on EHR data stored at another hospital server, then the server forwards the request to the remote hospital server using structured overlay network. All the read and write requests to the distributed database has to be authorized using Policy Decision Point (PDP) server. The server containing the EHR data will also log the object data to the local log-structured database after the authorization process.
2.2.3 Cloud distribution architecture

The nodes in the cloud database of the DIGHT are organized using structured overlay networks as shown in Fig: 2.5. Structured overlay networks are a class of decentralized distributed systems that provide a name-value lookup service similar to a hash table. Overlay network structure like a ring or a tree network can be used to organize the nodes. Overlay network provides the facility to find the data for the given key efficiently by routing requests over the network nodes. Responsibility for maintaining the mapping from keys to values is distributed among the nodes.
2.2.4 Implementation Language

DIGHT is implemented in Java using Kompics. Kompics supports the development of distributed systems as event-based systems. Kompics will enable us to build a network model that captures the network environment between Indian hospitals, and simulate our code using that network model. As Kompics supports decoupling of components, the same simulation code can be reused in the production code, helping improve our confidence in a correctly functioning system.
Chapter 3

Model/Implementation

3.1 Enhancements and New features of XACML 3.0

The XACML 3.0 specification has gone through many enhancements and some new features were also added. They are explained below.

Multiple Decision Profile  The Multiple Resource Request of XACML 2.0 is rechristened as Multiple Decision Profile. This profile allows the requester to ask multiple questions to PDP in the same request. The PDP answers to all those questions with a single response containing multiple decisions. This profile is very useful in web portal based scenarios. The communication overhead between PEP and PDP is also considerably reduced because of this.

Administration Profile  Administration profile of XACML 3.0 allows the global administrators to delegate their administrative rights to the local administrative bodies in a constrained manner. Delegation is very much useful in the federation scenarios, cloud based scenarios and in vast domains where the local knowledge is a must need to design the authorization policies. Delegation profile allows the global administrator to define policies which controls how the local administrator can delegate the administrative rights.

ObligationExpressions and AdviceExpressions  Both the features are new to XACML 3.0 which allows the PDP to add few statements to the response along with the decision of Permit or Deny. The PEP has to understand the Obligations returned before taking actions on the decision. If the PEP does not comply with the Obligations, it cannot enforce the decision on the requester. But the PEP can safely ignore the Advice.

Combining Algorithms  XACML combines the decisions from multiple policies and rules into a single decision. Policy Combining algorithms and Rule Combining
algorithms are used respectively to combine the decisions from multiple policies and rules. XACML 3.0 enhances the already existing combining algorithms and also introduces two new Policy Combining algorithms and Rule Combining algorithms.

**Functions and Datatypes** XACML 3.0 also introduces new datatypes and functions to apply on the attributes. Many String manipulation functions and XPath manipulation functions are introduced in XACML 3.0. The new functions give more control over the attributes and policy design. XACML is a rich language with many standard functions to handle almost all the use cases. Also, XACML allows the user to define their own functions, datatypes and combining algorithms.

### 3.2 Enhancements Done

The main aim of this thesis is to design and implement a prototype for distributed PEP and PDP system. The existing open source implementations of PDP complies with XACML 2.0 standards. There are no open source implementations that complies with XACML 3.0 standards. The XACML 3.0 standards provide many important features including delegation. For DIGHT project, the delegation was considered an important requirement. So, it was decided to implement XACML 3.0 and then design a prototype version to allow a doctor to register a patient, add a medical record and access the existing patient and medical record through SMS. The SMS request is then converted to XACML request and authorized using the PDP implementation.

After detailed analysis and research on the existing implementations Enterprise Java XACML [14], Sun’s XACML Implementation [8] and SICSACML [21], it was decided to use the SICSACML, a draft version of XACML 3.0 implementation by the researchers in SICS on top of Sun XACML 2.0 implementation. The Sun XACML 2.0 implementation is downloaded from [9] and SICSACML patch file downloaded from [22] is applied to that. The entire codebase is changed to use generics to provide type safety. The existing implementation read a request file, a list of policy files in xml format and stored them as Java objects. The XML parsing is complex and time consuming. It is also tough to add a policy on the fly. Hence, it was decided to parse the policies and store them in relational database systems. MySQL, an open source relational database implementation was selected to store the parsed policies. The schema is designed such that each element of XACML standard is stored in a table named according to the element name. Each attribute of the element has a column. Each row of the table is assigned an unique primary key value which auto increments for each row insert. The element is referenced in other tables using this primary key value. Each element within other element has a column in the containing element table and references the primary key from its appropriate table.
3.2. ENHANCEMENTS DONE

3.2.1 Policy Administration Point

Policy Administration Point is implemented to parse the policies. A single policy file or a directory containing policy files and a configuration file should be passed as arguments to the application. The configuration file contains the database connection information. The application parses the policy files and stores them into the relational database. If there are any syntax or semantic errors, the application will throw `ParseException`.

3.2.2 Policy Decision Point

Design

The SICSACML provides abstract class implementations for CombiningAlgorithms, Datatypes and Functions. These classes can be extended to add new Combining Algorithms, Datatypes and Functions. The XACML 3.0 standards introduced new Policy and Rule Combining Algorithms like Permit-Unless-Deny, Deny-Unless-Permit. The existing Policy and Rule Combining algorithms like Permit-Overrides, Deny-Overrides, Ordered-Permit-Override and Ordered-Deny-Override are enhanced in XACML 3.0 standards. New classes were introduced for all these combining algorithms. The old combining algorithms are still retained to achieve backwards-compatibility. Many new functions were also introduced in XACML 3.0 standards especially for string manipulation. Also, the optional Xpath functions were not supported in SICSACML implementation. New classes were added to support all these functions. A class for the new datatype XpathExpression was also added. New classes were introduced for XACML elements ObligationExpressions, AdviceExpressions, AttributeAssignmentExpressions and AttributeAssignment. The Policy evaluation is modified such that the ObligationExpressions and AdviceExpressions are evaluated when the decision matches the FulfillOn and AppliesTo attribute respectively. The evaluated ObligationExpression and AdviceExpression returns Obligation and Advice elements which are added to the XACML Response.

The XACML 3.0 Administration and Delegation profile [16] was also implemented. This profile controls the types of policies a user can create or modify. Dynamic delegation facilitates the user to create policies on the fly to delegate some of his authorities to other person under certain circumstances. For example, "while I am on vacation, Mary can approve requests". If the policy does not contain the trusted issuer, then the policy is reduced by creating the Administrative Request Context. Reduction is the process by which the authority of the policy is established. The process of creating Administrative Request and evaluating the reduction explained in [16] is implemented.
Implementation

Policy Decision Point takes an optional configuration file as argument. If the configuration file is not passed as argument, then it will check for the System property called com.sunxacml.PDPConfigFile. If the property is also not defined, then the application quits throwing ParsingException. If the file is provided, the configuration is read and the PDP system is initialized. The file contains the configuration settings about the PolicyFinderModules, AttributeFinderModules and various factory classes for Functions, Combining Algorithms and Attributes. If the Database-PolicyFinderModule is used, then the database connection information is also provided through this file. After the system is initialized, it will listen on the port specified in the configuration file for XACML Request. If the port is not specified, the system will listen on the default port value 5555.

The existing SICSACML codebase is designed such that each element in XACML standards has a corresponding class. Each class has a static method named getInstance(Node root) which parses the xml element from the policy file and returns the instance of the class. Since the policy elements are now stored in the database, a new static method is introduced in all those classes with signature createInstance (Integer id) to read from the database. The unique primary key value is passed as argument to the method. During PDP system initialization, it reads all the top level Policy and PolicySet from the database and store them as objects. Whenever a XACML Request comes, a new thread is spawned and the request is serviced.

Each thread will parse the request file from the socket and evaluate the request against the existing policies. The evaluation result along with the Obligations and Advices are encoded as the XACML Response file and sent through the socket.

3.2.3 Policy Enforcement Point prototype

Design

The prototype system design is shown in Fig: 3.1. The doctor will send a text sms from his/her registered mobile phone device. The SMS gateway will receive the text message and trigger the SmsRequest event. The Policy Enforcement Point (PEP) looks up the attribute database and validates the sender.

Based on the SmsRequest it creates the XACML Request. For creating the Request, it communicates DIGHT client to get the patient or medical record. The Request Content element is created with the information retrieved. Then, the XACML Request is transmitted to the PDP through socket and wait for the response. Once the XACML Response is received, it is processed. If the decision is not Permit, then appropriate error message is sent through sms. If the decision is Permit, then the obligations are processed if available. If the PEP cannot satisfy the obligations, it should not allow the requester to access the resource. If PEP can understand the obligations and service it, then the permission is given to DIGHT to modify the database with appropriate information. The SmsResponse is then
triggered with the proper response value returned from DIGHT client. The policy is found in Appendix A.1

Implementation

The prototype is also implemented using Kompics. The detailed tutorial on using Kompics is available at [2]. In this prototype implementation, three Components are defined SmsMain, PEPComponent and Dight. The relation between these components is described in Fig: 3.2.

The SmsMain component contains two ports Network and PepPort, Dight contains DightPort and Network, PEPComponent contains DightPort and PepPort. SmsMain is the main component which initializes the system and all the sub components. After system initialization, it starts a web server which waits to service the incoming text messages. SmsMain communicates with the PEPComponent through the PepPort. The event communicated between them are SmsRequest and SmsResponse. Dight component communicates with the PEPComponent through the DightPort. PEPComponent triggers the request event to enter new patient details, new medical record for an existing patient, get patient details or get a single medical record of a patient. Dight component triggers the corresponding response event. Dight component communicates with the MinaNetwork component through Network port. Dight component triggers write and read request to the DIGHT database through this port. It also provides the handler for the corresponding response events.

The prototype allows four different events.

- Add patient details
- Add medical record
- Get patient details
- Get medical record

Add patient details  The sequence diagram for adding a patient’s details is shown in Fig: 3.3. The sms request received from the doctor is parsed by the PEPComponent. Then the details about the doctor is retrieved from the attribute database based on the mobile number. If no such entry exists in the attribute database, an error response is sent. XACML Request object is created, serialized and then passed to the PDP through socket. The PDP returns Permit only if the sender’s role is ‘doctor’ and the action-id is ‘register’. Else, PDP sends Deny as the decision.

If the response contains the decision as Permit, then it triggers the EnterPatientDetailsReq event. If the database update is successful, then the PEPComponent receives EnterPatientDetailsResp event. The event handler triggers the SmsResponse event with the patient id generated by DIGHT. If any error happens during
the process, then $SmsResponse$ is triggered with the proper error message.

**Syntax**

**Request**

$?(m|f|o):(s|m|d|w):(yymmdd):(blood-type):(first-name):(middle-name):$

$(last-name):(address):(tel):(email)$

where

$(m|f|o)-(male|female|other),$

$(s|m|d|w)-single|married|divorced|widow,$

$blood-type- o-| o+| a-| a+| b-| b+| ab-| ab+$

$tel - +(country-code)-mobile-no$

**Response**

$?(Ok|Fail):(PatientId|FailureReason)$

---

**Add medical record**  The sequence diagram for adding a medical record for an existing patient is shown in Fig: 3.4. The sms request received from the doctor is parsed by the PEPComponent. Then the doctor’s details are retrieved from the attribute database based on the mobile number. If no such entry exists in the attribute database, an error response is sent. The requested patient details are retrieved from DIGHT by triggering $GetPatientDetailsReq$. If the patient does not exist, the error response is sent. If the response is received, then XACML Request object is created along with patient details, serialized and then passed to the PDP through socket. The PDP returns Permit only if the sender’s role is ‘doctor’, the action-id is ‘add-record’ and the subject-id is equal to the patient’s physician-id. Else, PDP sends Deny as the decision. If the decision is Permit, the policy also adds an Obligation in the response to send a sms to the patient regarding the access to his record.

If the response contains the decision as Permit, then it triggers the $EnterMedicalRecordReq$ event. If the database update is successful, then the PEPComponent receives $EnterMedicalRecordResp$ event. The event handler triggers the $SmsResponse$ event with the medical record id generated by DIGHT along with the patient id. If any error happens during the process, then $SmsResponse$ is triggered with the proper error message.

**Syntax**

**Request**

$!patient-id:(record-type):(body-part):(free-text-description of$

$problem/treatment)$

**Response**

$!(Ok|Fail):(RecordId|FailureReason)$
3.2. ENHANCEMENTS DONE

Get patient details  The sequence diagram for getting the details of an existing patient is shown in Fig: 3.5. The sms request received from the doctor is parsed by the PEPComponent. Then the details about the doctor is retrieved from the attribute database based on the mobile number. If no such entry exists in the attribute database, an error response is sent. The requested patient details are retrieved from DIGHT by triggering GetPatientDetailsReq. If the patient does not exist, the error response is sent. If the response is received, then XACML Request object is created along with patient details, serialized and then passed to the PDP through socket. The PDP returns Permit only if the sender’s role is ‘doctor’, the action-id is ‘read-patient’ and the subject-id is equal to the patient’s physician-id. Else, PDP sends Deny as the decision. If the decision is Permit, the policy also adds an Obligation in the response to send a sms to the patient regarding the access to his record.

If the response contains the decision as Permit, then it encodes the patient details in the proper format and triggers the SmsResponse event. If any error happens during the process, then SmsResponse is triggered with the proper error message.

Syntax

Request
@patient-id

Response
@(fail:failure reason) |(m|f|o):(s|m|d|w):(yymmdd):(blood-type):
(first-name):(middle-name):(last-name):(address):(tel):(email)

Get medical record  The sequence diagram for adding a medical record for an existing patient is shown in Fig: 3.6. The sms request received from the doctor is parsed by the PEPComponent. Then the details about the doctor is retrieved from the attribute database based on the mobile number. If no such entry exists in the attribute database, an error response is sent. The requested patient details are retrieved from DIGHT by triggering GetPatientDetailsReq. If the patient does not exist, the error response is sent. If the response is received, then XACML Request object is created along with patient details, serialized and then passed to the PDP through socket. The PDP returns Permit only if the sender’s role is ‘doctor’, the action-id is ‘add-record’ and the subject-id is equal to the patient’s physician-id. Else, PDP sends Deny as the decision. If the decision is Permit, the policy also adds an Obligation in the response to send a sms to the patient regarding the access to his record. If the response contains the decision as Permit, then it triggers the GetMedicalRecordReq event. If the database update is successful, then the PEPComponent receives GetMedicalRecordResp event. The event handler triggers the SmsResponse event with the proper encoded medical record message. If any error happens during the process, then SmsResponse is triggered with the proper error message.
Syntax

Request
,patient-id:record-id

Response
,(fail:failure reason) | (record-type):(body-part):(free-text-description of problem/treatment)
3.2. ENHANCEMENTS DONE
Figure 3.2. PEP prototype design
Figure 3.3. Sequence diagram for adding patient details
Figure 3.5. Sequence diagram for retrieving Patient Details
Figure 3.6. Sequence diagram for retrieving a medical record.
Chapter 4

Evaluation

There is no open source implementation of XACML 3.0 specifications. There are open source implementation for XACML 2.0 by Sun Microsystems and few other people. This is the first open source implementation for XACML 3.0. XACML 3.0 specifications introduces many new features (Refer Section 3.1) which are very critical for Electronic Healthcare systems. For example, delegation of authorities dynamically is an important feature required in healthcare environment. XACML 3.0 also overcomes some of the constraints put by XACML 2.0 on the target attributes. All this features and enhancements are implemented in this project.

4.1 Prototype Evaluation

This section evaluates the prototype implementation. It first describes the environment used followed by the different inputs and their corresponding outputs for all the four use cases.

4.1.1 Environment Setup

The prototype uses a policy (Refer Appendix A.1) which gives Permit under the following conditions:

- The attribute urn:sics:se:dight:xacml:names:attribute:role is doctor
- The attribute urn:sics:se:dight:xacml:names:attribute:database-name is patient-demographics
- The attribute urn:oasis:names:tc:xacml:1.0:action:action-id is add-patient or any of add-record, read and the attribute
urn:sics:se:dight:xacml:names:attribute:physician-id should be equal to the physician id field of the patient record in the Request context.

The attribute database contains a single table which contains the information about all the registered users. The table has the primary key as the mobile number. It contains the role of the user, registration number of the user and his personal details. For evaluation, the sample contents used is given in Table :4.1

<table>
<thead>
<tr>
<th>Mobile Num</th>
<th>First Name</th>
<th>Last Name</th>
<th>Role</th>
<th>Reg Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>+46707765274</td>
<td>Saravanakumar</td>
<td>Sankaramoorthy</td>
<td>Doctor</td>
<td>317142</td>
</tr>
<tr>
<td>+46732035012</td>
<td>Haresh</td>
<td>Rajendran</td>
<td>Specialist</td>
<td>317150</td>
</tr>
<tr>
<td>+46707765723</td>
<td>Jim</td>
<td>Dowling</td>
<td>Doctor</td>
<td>876555</td>
</tr>
</tbody>
</table>

### 4.1.2 Add Patient Details

The following are the few possible cases for adding a new patient’s details to the database.

**Case 1**

The below request is sent from an authorized user. The PEP fetches the information about the user from the attribute database and form the XACML request and send it to the PDP. The PDP sends the XACML response with Permit decision. The PEP contacts the DIGHT to update the database with the patient details and gets the PatientId. It then triggers the response to the sender with the PatientId.

**Request**

Request is sent from +46707765274.


**Response**

?Ok:16059

**Case 2**

The syntax is wrong in the below request. So the PEP will not send the request to PDP. It triggers the error response to the sender.

**Request**

The request is sent from +46707765274.


**Response**

?Fail:Request syntax is wrong
Case 3

In this case the response is sent from an unauthorized user. The people with role as 'Doctor' in the Table :4.1 are the authorized users. Hence the PDP sends the response with Deny decision. The PEP triggers the error response to the sender.

Request  Request is sent from +46732035012.
carol.alice@domain.com

Response  ?Fail:access denied

4.1.3 Add Medical Record

Below are few of the possible cases for adding medical record to a patient.

Case 1

The request is sent from the doctor who is responsible for the particular patient. The PEP fetches the attributes of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Permit decision since the sender is the responsible doctor for the patient to whom the record is added. The PEP parses the response, fulfils the obligation, trigger the request to DIGHT to update the database and then sends the response with the RecordId to the sender.

Request  The Request is sent from +46707765274.
!16059:xray:Chest: Diagnosis glandular fever. Treated with antibiotics

Response  !0k:8766

Case 2

The request is sent from an authorized user. So the PEP fetches the attribute of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Deny decision since the sender is not the responsible doctor for the patient to whom the record is added. The PEP parses the response and triggers the error response to the sender.

Request  The Request is sent from +46707765273.
!16059:xray:Chest: Diagnosis glandular fever. Treated with antibiotics

Response  !Fail:access denied
Case 3
The request is sent from an unauthorized user. The PEP tries to fetch the attribute of the sender. Since the sender information is not available in the attribute database, it triggers the error response to the sender.

Request  The Request is sent from +46707700000.
!16059:xray:Chest: Diagnosis glandular fever.
Treated with antibiotics

Response  !Fail:access denied

Case 4
The request is sent from an authorized user but the PatientId does not exist. The PEP fetches the attribute of the sender, tries to form the XACML request. Since the PatientId is not available in the DIGHT database, it triggers the error response to the sender.

Request  The Request is sent from +46707700000.
!16000:xray:Chest: Diagnosis glandular fever.
Treated with antibiotics

Response  !Fail:Patient does not exist

Case 5
The syntax is wrong in the below request. So the PEP will not send the request to PDP. It triggers the error response to the sender.

Request  The Request is sent from +46707700000.
!16000:Chest: Diagnosis glandular fever.
Treated with antibiotics

Response  !Fail:Invalid request syntax

4.1.4 Get Patient Details
The following are the few possible cases while getting a patient’s details from the database.
Case 1

The request is sent from the doctor who is responsible for the particular patient. The PEP fetches the attributes of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Permit decision since the sender is the responsible doctor for the patient whose details is requested. The PEP parses the response, fulfills the obligation, trigger the request to DIGHT to retrieve from the database and then sends the response with the patient details to the sender.

Request  Request is sent from +46707765274.
@16059

Response  @m:m:730531:b+:Alice:Bob:Carol:Stockholm, Sweden:
+46732501234: carol.alice@domain.com

Case 2

The request is sent from an authorized user. So the PEP fetches the attribute of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Deny decision since the sender is not the responsible doctor for the patient whose details is requested. The PEP parses the response and triggers the error response to the sender.

Request  The Request is sent from +46707765273.
@16059

Response  @Fail:access denied

Case 3

The request is sent from an unauthorized user. The PEP tries to fetch the attribute of the sender. Since the sender information is not available in the attribute database, it triggers the error response to the sender.

Request  The Request is sent from +46707700000.
@16059

Response  @Fail:access denied

Case 4

The request is sent from an authorized user but the PatientId does not exist. The PEP fetches the attribute of the sender, tries to form the XACML request. Since
the PatientId is not available in the DIGHT database, it triggers the error response to the sender.

**Request**  The Request is sent from +46707765274.
@16000

**Response**  @Fail:Patient does not exist

**Case 5**

The syntax is wrong in the below request. So the PEP will not send the request to PDP. It triggers the error response to the sender.

**Request**  The Request is sent from +46707700000.
@name

**Response**  @Fail:Invalid request syntax

### 4.1.5 Get Medical Record

Below are few of the possible cases for getting the medical record of a patient.

**Case 1**

The request is sent from the doctor who is responsible for the particular patient. The PEP fetches the attributes of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Permit decision since the sender is the responsible doctor for the patient whose details is requested. The PEP parses the response, fulfils the obligation, trigger the request to DIGHT to retrieve from the database and then sends the response with the medical record details to the sender.

**Request**  Request is sent from +46707765274.
,16059:8766

**Response**  ,xray:Chest: Diagnosis glandular fever. Treated with antibiotics

**Case 2**

The request is sent from an authorized user. So the PEP fetches the attribute of the sender, forms the XACML request and sends it to the PDP. The PDP sends the response with Deny decision since the sender is not the responsible doctor for
the patient whose details is requested. The PEP parses the response and triggers the error response to the sender.

**Request**  The Request is sent from +46707765273.
,16059:8766

**Response**  ,Fail:access denied

**Case 3**
The request is sent from an unauthorized user. The PEP tries to fetch the attribute of the sender. Since the sender information is not available in the attribute database, it triggers the error response to the sender.

**Request**  The Request is sent from +46707700000.
,16059:8766

**Response**  ,Fail:access denied

**Case 4**
The request is sent from an authorized user but the PatientId does not exist. The PEP fetches the attribute of the sender, tries to form the XACML request. Since the PatientId is not available in the DIGHT database, it triggers the error response to the sender.

**Request**  The Request is sent from +46707765274.
,16000:8766

**Response**  ,Fail:Patient does not exist

**Case 5**
The request is sent from an authorized user but the RecordId does not exist. The PEP fetches the attribute of the sender, tries to form the XACML request. Since the RecordId is not available in the DIGHT database, it triggers the error response to the sender.

**Request**  The Request is sent from +46707765274.
,16059:8777

**Response**  ,Fail:Medical record does not exist
Case 6

The syntax is wrong in the below request. So the PEP will not send the request to PDP. It triggers the error response to the sender.

Request  The Request is sent from +467077000000, 16059

Response  Fail: Invalid request syntax

4.2 Performance Evaluation

4.2.1 DIGHT database Setup

The DIGHT database is maintained in seven different nodes in the cloud. The network address and web address mapping to the peer is given in Table: 4.2. The configuration of the nodes are listed below in Table: 4.3.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Network Address</th>
<th>Web Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td><a href="mailto:0@cloud1.sics.se">0@cloud1.sics.se</a>:50626</td>
<td><a href="http://193.10.64.107:50625/0/CATS">http://193.10.64.107:50625/0/CATS</a></td>
</tr>
<tr>
<td>1000</td>
<td><a href="mailto:0@cloud2.sics.se">0@cloud2.sics.se</a>:60424</td>
<td><a href="http://193.10.64.85:60423/0/CATS">http://193.10.64.85:60423/0/CATS</a></td>
</tr>
<tr>
<td>10000</td>
<td><a href="mailto:0@cloud3.sics.se">0@cloud3.sics.se</a>:49162</td>
<td><a href="http://193.10.64.86:49161/0/CATS">http://193.10.64.86:49161/0/CATS</a></td>
</tr>
<tr>
<td>20000</td>
<td><a href="mailto:0@cloud4.sics.se">0@cloud4.sics.se</a>:27683</td>
<td><a href="http://193.10.64.109:27682/0/CATS">http://193.10.64.109:27682/0/CATS</a></td>
</tr>
<tr>
<td>30000</td>
<td><a href="mailto:0@cloud5.sics.se">0@cloud5.sics.se</a>:6204</td>
<td><a href="http://193.10.64.116:6203/0/CATS">http://193.10.64.116:6203/0/CATS</a></td>
</tr>
<tr>
<td>40000</td>
<td><a href="mailto:0@cloud6.sics.se">0@cloud6.sics.se</a>:47725</td>
<td><a href="http://193.10.64.200:47724/0/CATS">http://193.10.64.200:47724/0/CATS</a></td>
</tr>
<tr>
<td>50000</td>
<td><a href="mailto:0@cloud7.sics.se">0@cloud7.sics.se</a>:26246</td>
<td><a href="http://193.10.64.216:26245/0/CATS">http://193.10.64.216:26245/0/CATS</a></td>
</tr>
</tbody>
</table>

Table 4.3. Node configuration in cloud

<table>
<thead>
<tr>
<th>CPU</th>
<th>6-core 64 bit AMD Opteron</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>Linux 2.6.38-8-server</td>
</tr>
<tr>
<td>Memory</td>
<td>2040.14 MB</td>
</tr>
</tbody>
</table>

The Table: 4.4 shows the key range and the peers which contains the replica of the data. The peers are arranged in the form of a ring. Each data is always replicated and stored in three peers. For example, if the key value is 678, then the data for the key is stored in the peers 10000, 20000 and 30000. By replication of the data, we assure that the system works even when there is partial failure. If the Peer 1000 fails, then the data is replicated to one more peer in the ring.
4.2. PERFORMANCE EVALUATION

<table>
<thead>
<tr>
<th>Key Range</th>
<th>Replication Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>(100, 1000]</td>
<td>10000, 20000, 30000</td>
</tr>
<tr>
<td>(1000, 10000]</td>
<td>20000, 30000, 40000</td>
</tr>
<tr>
<td>(10000, 20000]</td>
<td>30000, 40000, 50000</td>
</tr>
<tr>
<td>(20000, 30000]</td>
<td>40000, 50000, 100</td>
</tr>
<tr>
<td>(30000, 40000]</td>
<td>50000, 100, 1000</td>
</tr>
<tr>
<td>(40000, 50000]</td>
<td>100, 1000, 10000</td>
</tr>
<tr>
<td>(50000, 100]</td>
<td>1000, 10000, 20000</td>
</tr>
</tbody>
</table>

4.2.2 Evaluation

The PDP and PEP run on the same machine. The configuration of the machine is given in Table: 4.5.

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel Core 2 Duo</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>Linux 2.6.31-14-generic</td>
</tr>
<tr>
<td>Memory</td>
<td>2894 MB</td>
</tr>
<tr>
<td>JRE</td>
<td>OpenJDK Runtime Environment 6</td>
</tr>
</tbody>
</table>

The XACML standards does not specify how the communication between PEP and PDP should happen. We have chosen sockets as means of communication. The XACML Request and Response are transmitted as serialized objects between the PDP and PEP using sockets. But this is not using any authentication strategy. Hence it is not secure. But there are many other ways to make this communication safe and secure. One of them is discussed as future work using SAML. The response time taken by PDP to evaluate the XACML Request is approximately 1ms. The turnaround time taken for a complete request varies on the mobile service provider. The turnaround time taken for the PEP to handle the SMS request is approximately 5ms. This can be largely reduced if SAML profile for XACML is implemented. The PEP communicates with the DIGHT database using Webservice calls for reading and writing. The performance values for various parts of the system is given in Table: 4.6.

<table>
<thead>
<tr>
<th>Performance Values</th>
<th>~1ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response time of PDP</td>
<td>~1ms</td>
</tr>
<tr>
<td>DIGHT read</td>
<td>~0.65ms</td>
</tr>
<tr>
<td>DIGHT write</td>
<td>~0.70 ms</td>
</tr>
<tr>
<td>Turnaround time of PEP once sms</td>
<td>~5ms</td>
</tr>
<tr>
<td>request is received</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion and Future Work

5.1 Conclusion

With computerization of healthcare systems increasing widely, providing access to sensitive information is one of the major areas of concern. Extensible authorization using XACML provides such solution in a secured, standardized manner. The idea is to distribute the policy authorization system in the cloud. The hospitals spread over India have their own personalized database. This thesis which is part of DIGHT project, provides the solution to distribute the database using key-value pair architecture. A copy of the private database from the hospitals is stored in the cloud and duplicated in the nodes in a geographically distributed way. The thesis implemented XACML 3.0 specification on top of the SICSACML implementation. This is the first open source implementation of XACML 3.0 specifications. A prototype is developed to authorize the requester to access the distributed DIGHT database through sms. The prototype was demonstrated at the SICS open house event in early May, 2011. The detailed explanation about the prototype is mentioned in the Chapter 3 of this report.

Another goal was to distribute the Policy Decision Point and to store the authorization policies in the relational database systems. The parsing of the policy from the xml files is very time consuming and also insecure. By storing the policy in database clusters, it can easily be replicated, accessed in a fast and secured way. The prototype implementation reads the policy from the mysql database. The policy database is then distributed with mysql clusters and the performance evaluation is done. The detailed explanation about the performance variants is mentioned in the evaluation chapter of the report.

5.2 Future Work

Policy Information Point (PIP) In the prototype implementation, the attributes related to the sender is obtained from a relational database system by
the PEP. But XACML standards suggest to use Policy Information Point to get
the value for attributes using key values. The PDP will request for the required
attributes to the PIP. The SICSACML implementation provides an abstract class
which can be extended to provide PIP implementation. The PIP implementation
can use LDAP directory, database system or even file system to map attributes. The
possible future work is to use PIP to fetch attributes from the DIGHT database.

**SAML Assertion profile for XACML**  The XACML standards does not spec-
ify how the PEP and PDP communicates. In the prototype implementation, TCP
socket has been used for communication. Security Assertion Markup Language
(SAML) [13] is a xml-based open standards from OASIS for exchanging authentica-
tion and authorization information between identity provider and security provider.
In the SAML 2.0 core specification, it is mentioned that the SAML AuthzDecision-
Statement is "frozen" and it encouraged people to consider XACML. The XACML
Request and Response can be sent through SAML Assertion [15] messages. There
is a profile of SAML 2.0 which specifies how to protect, transport and request
XACML schema instances and other information needed by XACML implementa-
tion using SAML. The design, implementation and testing of this profile for the
XACML implementation could be the other potential work.

**Domain Specific Language for XACML**  Policy Authoring is a very com-
plex job. XACML policies are written using XML. Hence it is not easy to read
and understand. The lengthy URI’s for the attributes, datatypes, functions and
algorithms makes it error prone. So if the policy language can be simplified by
introducing a new language which hides the complexity of the XACML policies. A
small step has been taken towards this in [5][1]. This can be further explored and
extended for XACML 3.0. The work done is on XACML 2.0.
Appendix A

Samples

A.1 Policy Used

```xml
<?xml version="1.0" encoding='UTF-8'?>
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
       xmlns:xacml="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
       http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
       PolicyId="urn:sics:se:dight:xacml:names:policyid:doctorpolicy"
       Version='1.0'
       RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-
       algorithm:permit-overrides">
  <Description>Policy for any doctor accessing the patient database</Description>
  <PolicyDefaults>
    <XPathVersion>http://www.w3.org/TR/1999/Rec-xpath-19991116</XPathVersion>
  </PolicyDefaults>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
          <AttributeValue DataTime='http://www.w3.org/2001/XMLSchema#string'>doctor</AttributeValue>
          <AttributeDesignator MustBePresent='false'
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
            AttributeId="urn:sics:se:dight:xacml:names:attribute:role"
            DataType='http://www.w3.org/2001/XMLSchema#string' />
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</Policy>
```
<AnyOf>
<AllOf>
<Match MatchId="urn:oasis:names:tc:xacml:1.0:
:function:anyURI-equal">  
<AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#anyURI">jdbc:mysql://cloud3.sics.se:3307/</AttributeValue>  
<AttributeDesignator MustBePresent="false"
Category="urn:oasis:names:tc:xacml:3.0:
:attribute-category:resource"
AttributeId="urn:sics:se:dight:xacml:
:names:attribute:database-url"  
DataType="http://www.w3.org/2001/XMLSchema#
anyURI"/>
</Match>
</AllOf>
</AnyOf>
</Rule>
</Target>

<Rule RuleId="urn:sics:se:dight:xacml:
:names:ruleid:patient-access">
Effect="Permit">
Description>A physician may write any medical element in a
record for which he or she is the designated primary care
physician, provided an email is sent to the patient</Description>
</Target>
</AnyOf>
</AllOf>
</AnyOf>
</Rule>

<Rule RuleId="urn:oasis:names:tc:xacml:3.0:
:function:string-equal-ignore-case">  
<AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">patient-demographics</AttributeValue>  
<AttributeDesignator MustBePresent="false"
Category="urn:oasis:names:tc:xacml:3.0:
:attribute-category:resource"
AttributeId="urn:sics:se:dight:xacml:
:names:attribute:database-name"  
DataType="http://www.w3.org/2001/XMLSchema#
string"/>
</Match>
</AllOf>
</AnyOf>
</AnyOf>
</AllOf>
</AnyOf>
</Rule>

<Rule RuleId="urn:oasis:names:tc:xacml:3.0:
:function:string-equal-ignore-case">
<Rule RuleId="urn:sics:se:dight:xacml:names:ruleid:patient-update">
  <Effect>Permit</Effect>
  <Description>A doctor may update any medical element or patient demographics for which he or she is the designated primary care physician, provided an sms is sent to the patient</Description>
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">patient-demographics</AttributeValue>
          <AttributeDesignator MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
            AttributeId="urn:sics:se:dight:xacml:names:attribute:database-name"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
        </Match>
      </AllOf>
    </AnyOf>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">update</AttributeValue>
          <AttributeDesignator MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
            AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
            DataType="http://www.w3.org/2001/XMLSchema#string" />
        </Match>
      </AllOf>
    </AnyOf>
  </Target>
</Rule>
<Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">add-record</AttributeValue>
</Match>

<Match MatchId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
    <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">read</AttributeValue>
</Match>

<Condition>
    <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
            <AttributeDesignator MustBePresent="false"
                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
            <AttributeSelector MustBePresent="false"
                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                Path="/record/physician-id/text()"
                DataType="http://www.w3.org/2001/XMLSchema#string"/>
        </Apply>
    </Apply>
</Condition>
A.2 Sample Request

```xml
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource">
    <Attribute AttributeId="urn:sics:se:dight:xacml:names:attribute:database-url">
      <AttributeValue Data_Type="http://www.w3.org/2001/XMLSchema#anyURI">jdbc:mysql://localhost:3306/</AttributeValue>
    </Attribute>
  </Attributes>
</Request>
```
A.3 Sample Response

```xml
<Response xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17">
  <Result>
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
  </Result>
</Response>
```
Bibliography


