Chapter 5

Security Policy Framework

This chapter presents the implemented security policy framework. The key security requirements and policies of grid systems have been categorized under four heads namely authentication, privacy, trust and authorization. The previous chapter has explained different types of access control policies and the mechanisms used for their specification. This chapter presents models for handling each of the security requirements and policies related to authentication, privacy, trust and authorization together with their integration. Sections 5.1 to 5.3 present the individual authentication, privacy and trust models. Section 5.4 presents the policy based authorization framework and also discusses the integration of these models.

5.1 Authentication Model

Authentication system can be defined as a system that takes care of authentication related issues like single sign-on, delegation, nonrepudiation, secure logging, management of multiple user credentials *etc.* It is a detailed description of all aspects of a system that relate to authentication. Following sections describe how these features have been incorporated in the framework.

5.1.1 Single Sign-on and Delegation

The main problems related to authentication are of single sign-on and delegation. Single sign-on and delegation features depend on the type of security credentials used for authentication. *e.g.* these features can be easily provided, if the subject uses X.509 certificates, but, can not be provided if the subject is using username: password for

authentication. These features have been implemented in the framework through X.509 proxy certificates. Subjects are issued normal certificates by certificate authorities. Certificates bind subject's DN (Distinguished Name) to its public and private key. Public key is written in the certificate whereas private key remains with the subject. In order for subjects to prove their identity they must have i) Certificate and ii) Private Key. Authentication process involves presenting the certificate and proving the possession of the private key. Private key is held by subjects and is not disclosed to anyone under any circumstances. Proxy certificates differ from normal X.509 certificates in the sense that these are created by subjects and sent to other subjects/services so that they can act on original subject's behalf. Figure 5.1 shows how proxy certificates are created and used by a proxy service to act on original subject's behalf.

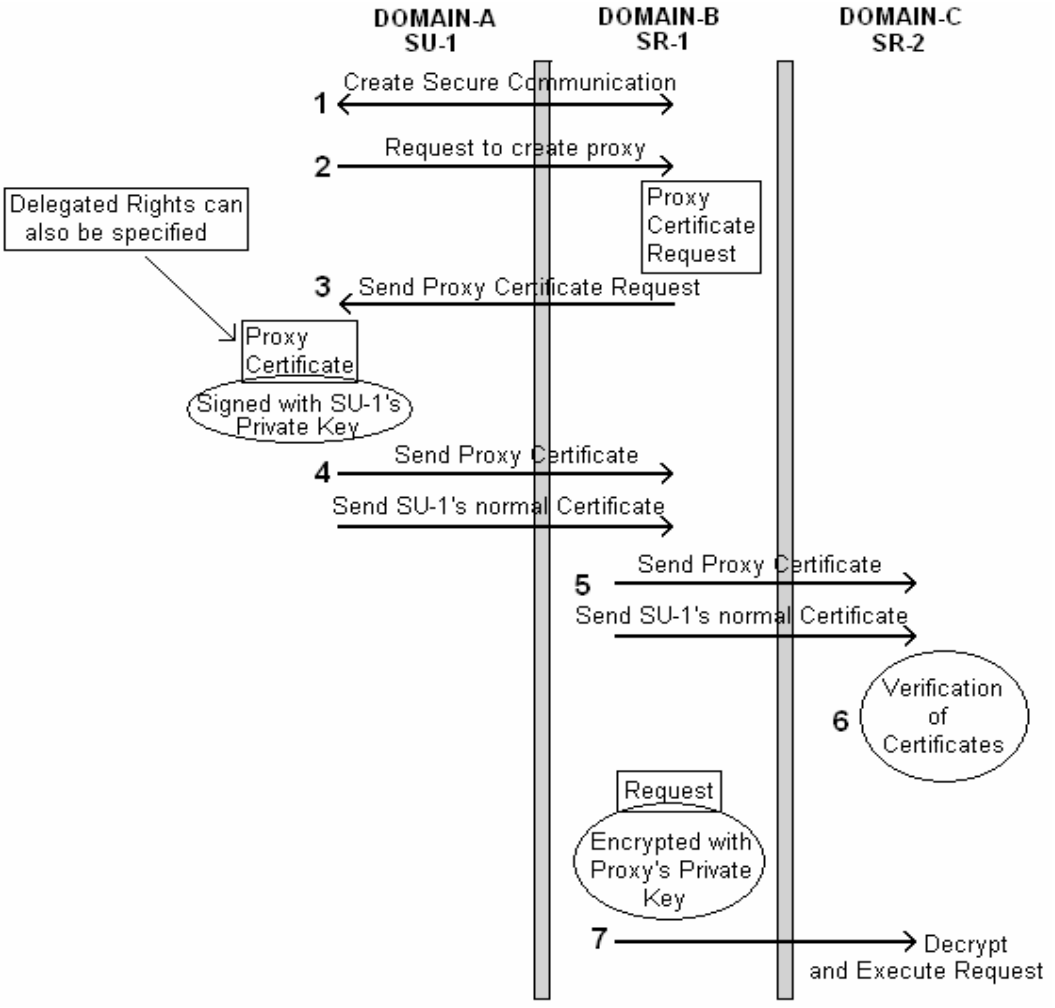

Figure 5.1: Proxy creation and action

Figure 5.1 shows SU-1 creating a proxy certificate and sending it to SR-1 so that SR-1 can act on its behalf. SR-1 can use this certificate to get authentication at other site on SU-1's behalf. Table 5.1 briefly explains the sequence of steps shown in Figure 5.1.

Step 1: A secure communication path is created between Subject SU-1 and the Service SR-1. Here SU-1 wants SR-1 to act on his behalf.
Step 2: SU-1 sends a request to create "proxy certificate request" to SR-1.
Step 3: A new key pair consisting of a public and private key is generated for use in proxy certificate. Using this public key, SR-1 creates a "proxy certificate request" and sends it to SU-1.
Step 4: SU-1 signs this "proxy certificate request" using his private key to create a proxy certificate. SU-1 then sends this proxy certificate to SR-1 along with his normal certificate.
Step 5: SR-1 sends SU-1's normal certificate and proxy certificate to other Service SR-2 to authenticate himself to SR-2 on SU-1's behalf. SR-1 is now acting as SU-1's proxy.
Step 6: SR-2 can verify the proxy certificate as it has SU-1's normal certificate, SU-1's public key (available through SU-1's normal certificate) and proxy certificate (signed using SU-1's private key).
Step 7: SR-1 can now send request to SR-2 on SU-1's behalf by encrypting that request with proxy certificate's private key. SR-2 can easily decrypt it as it has the proxy certificate's public key.

Table 5.1: Sequence of steps for proxy certificate creation and usage

In the above interactions, SU-1 uses his private key only once (while creating proxy certificate) and SR-1 is able to authenticate himself to SR-2 on SU-1's behalf. Delegation is also possible through proxy certificates. The delegated rights are expressed in XML and this information is attached to proxy certificates through PCI (Policy Certificate Information) field [63]. Subjects explicitly specify the subset of their total rights (which is to be delegated) in XML and this information is embedded in proxy certificates through PCI. Thus it is possible to support different delegation levels (*e.g.* impersonation, restricted delegation *etc.*) using proxy certificates. Step 3 in Figure 5.1 shows SU-1 specifying delegated rights in proxy certificate.

The authentication framework also supports trust based delegation in which rights are delegated only if trust with the target service/subject is greater than the threshold value. To calculate trust value, the pseudo codes described in Table 5.7, Table 5.8 and Table 5.9 are used. These are explained in Section 5.3.

5.1.2 Nonrepudiation, Confidentiality, Integrity and Secure Communication

Proxy certificates enable us to implement non repudiation also. As the subjects are digitally signing the certificates, they cannot deny from having sent the access request, as signature is possible only through their private key. Other security requirements like confidentiality and integrity of messages are guaranteed through encryption and signature techniques. For these, we are using XML-Encryption and XML-Signature features provided by .NET framework. For secure communication, we are making use of WS-SecureConversation specification implemented by WSE 3.0 toolkit under .NET environment. As already a lot of work has been done in the areas of confidentiality, integrity and secure communication, we are making use of existing technologies for these features. In addition to creation of proxy certificates, the implementation also supports the creation of custom security tokens and anonymous security tokens. Their purpose and use is explained in Section 5.2 while discussing privacy model.

5.1.3 Credential Management

As already mentioned, in grid, security requirements of different services can be different. *e.g.* In Domain A, Service-1 may require username/password, Service-2 may require Kerberos tickets and Service-3 may require X.509 certificates to authenticate subjects and so on. This is because services in a domain are provided by different service providers who generally belong to different physical organizations and they are free to implement any authentication mechanism of their choice to protect their services/resources. It is certainly neither possible nor a good idea to force different service providers to use same kind of security credentials to authenticate subjects. Though it may have advantages in

implementing a more robust authentication system but then the very purpose of integrating heterogeneous environments is lost.

Given this scenario, the main problem for the users is to satisfy the security requirements of different services. There are two approaches to the solution of this problem. First, we can provide a service to convert security credentials of one type to security credentials of other type. This approach looks fine as a first attempt but fails when we come across two incompatible credentials. *e.g.* it may be possible to convert a Kerberos ticket to equivalent X.509 certificate but it is certainly not possible to convert username: password to X.509 certificate or Kerberos Ticket. Therefore, we are following a different approach here. We want the subject to have different credentials to access different services. If a subject has to access different services from different domains then he must have all those types of security credentials as demanded by those services. One of these can be provided to the service according to the requirements of that service. Clearly, the second approach seems to be more suitable. But with this requirement, subjects will end up having many security credentials. Forcing users to manage credentials manually for each service is a tedious, error-prone and insecure task. The main problem here is the management of multiple user credentials.

As a solution to these problems, we propose grid domains to have a service to store, supply and maintain multiple security credentials for all of their subjects. The service will be responsible for managing subjects' multiple credentials and will be provided by the domain to which the subject belongs. In the implemented model, this service has been called Credential Manager Service (CMS). Subjects store all of their security credentials (X.509 certificates, Kerberos tickets, username: password pair *etc.*) in CMS.

To access a service, subject first discovers the security requirements of that service. Then subject checks the required security credentials in CMS. If the security credentials are present in CMS, then subject fetches those credentials from CMS by authenticating himself to CMS. If the required credentials are not in CMS, then subjects obtain them from the relevant certificate authority or from any other third party. The obtained credentials are then used by the subject to access the service. The new credentials are also stored in the CMS for future use. Figure 5.2 shows the usage of CMS where a Subject SU first discovers the security credentials required by Service SR using a discovery service. Subject then authenticates himself to CMS using username: password and fetches X.509 certificate. This certificate is then used by the Subject SU to access Service SR.

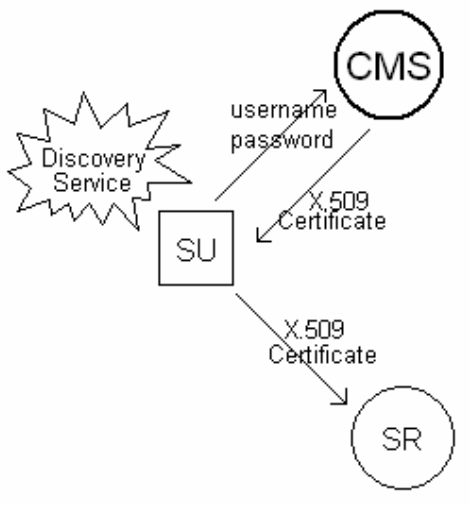

Figure 5.2: Credential Retrieval using Credential Manager Service

Following steps represent how a subject retrieves credentials using CMS.

Step 1: Subject discovers the security requirements of the service.
Step 2: Subject authenticates himself to CMS using the chosen credentials/mechanism.
Step 3: If authentication succeeds, subject checks CMS for the required credentials and performs the following:
a) If credentials are present, subject fetches them and goes to step 4.
b) If credentials are not present then subject obtains them from appropriate certificate authority.
c) Subject stores the credentials obtained from certificate authority in CMS for future use.
Step 4: Subject accesses the service using fetched credentials.

Table 5.2: Sequence of steps representing how a subject retrieves credentials using CMS

Figure 5.3 shows another example usage of CMS representing single sign-on and delegation also. A subject accesses CMS through username: password and CMS in turn, returns either X.509 Certificate/Kerberos Ticket as demanded by the subject. These credentials are passed to subject as SOAP messages. This interaction makes use of WS-Security specification. Subject then provides these credentials to Service SR-1 to access it. SU and SR-1 can also create a proxy certificate with/without right delegation to enable

SR-1 to access SR-2 on SU's behalf. Figure 5.3 also shows SR-1 passing proxy credentials to SR-2 to achieve single sign-on and delegation requirements.

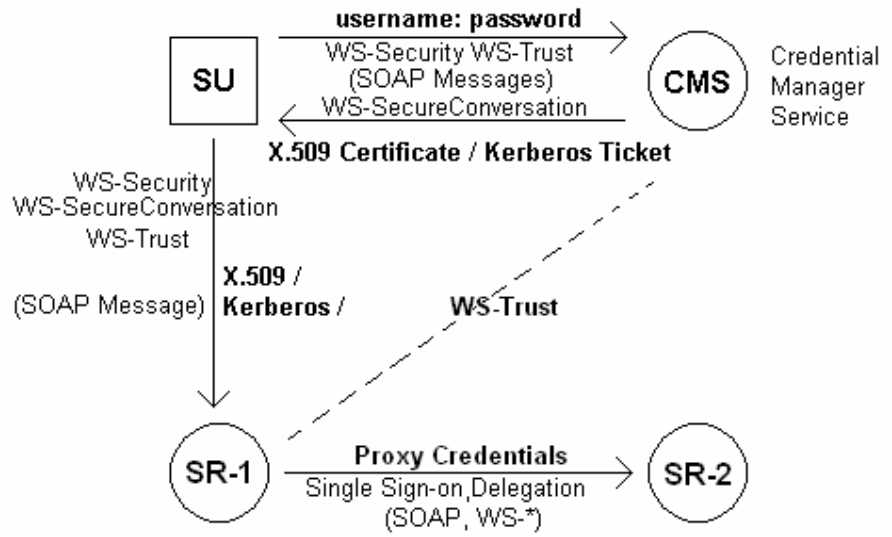

Figure 5.3: Usage of Credential Manager Service

Proxy credentials can also be stored in CMS. CMS can be used to store subject's private key also but in that case access to CMS will be through X.509 certificate and private key file is protected by password known only to the subject who owns the private key. If private key file is not stored in CMS then access to CMS can be provided through username: password also.

5.1.4 Services defined, implemented and exposed by Authentication Model

Authentication model has been implemented as a collection of authentication related services. Some of the important services defined, implemented and exposed by the authentication model are listed below:

Service Name	**Purpose**
upUpload( ):	To upload username: password pair.
x509Upload( ):	To upload X.509 certificate.
kerberosUpload( ):	To upload Kerberos security credential.

customUpload( ):	To upload custom security credential.
anonymousUpload( ):	To upload anonymous security credential.
proxyUpload( ):	To upload proxy certificate.
createProxy( ):	To create a proxy certificate.
searchCredential( ):	To search a particular credential from repository.
getup( ):	To retrieve username: password pair from repository.
getX509( ):	To retrieve X.509 certificate from repository.
getKerberos( ):	To retrieve Kerberos security credential from repository.
getCustom( ):	To retrieve custom security token from repository.
getAnonymous( ):	To retrieve anonymous security token from repository.
getProxy( ):	To retrieve proxy certificate from repository.
getAuthenticationPolicy( )	To retrieve authentication policy associated with a service.
setAuthenticationPolicy( )	To set authentication policy associated with a service.
removeUP( ):	To remove username: password pair from repository.
removeX509( ):	To remove X.509 certificate from repository.
removeKerberos( ):	To remove Kerberos security credential from repository.
removeCustom( ):	To remove custom security credential from repository.
removeAnonymous( ):	To remove anonymous security credential from repository.
removeProxy( ):	To remove proxy certificate from repository.
verifyUP( ):	To verify username: password pair.
verifyX509( ):	To verify X.509 certificate.
verifyKerberos( ):	To verify Kerberos security credential.
verifyCustom( ):	To verify custom security credential.
verifyAnonymous( ):	To verify anonymous security credential.
verifyProxy( ):	To verify proxy certificate.

Table 5.3: Services defined, implemented and exposed by authentication model

All these services have been implemented as web services in .NET environment. Table 5.3 lists only the service name and its purpose. The return type and arguments are not mentioned as most of the services exposed are overloaded. *e.g.* upUpload() service can accept two string values representing username and password for upload, as well as a

file (can be xml file also) containing username and password for upload. Similarly using searchCredential( ) service, subject can search for X.509 Certificates/Kerberos Tickets/Anonymous Security Credentials or any other type of credential depending on the arguments supplied to searchCredential( ) service. Moreover the services listed here are only the subset of the total services exposed by the authentication model. These services make use of other low level and high level services to achieve their tasks.

5.2 Privacy Model

A Privacy Model can be defined as a system that allows service requesters and service providers to state, evaluate and enforce their privacy requirements. It is a detailed description of all aspects of a system that relate to privacy. A privacy based authorization system grants specific type of access to specific requesters based on their authentication, what services/resources they are accessing, current state of the system and their conformance to established privacy policies. Privacy Model should be integrated with the authorization framework to provide privacy based access to services. Following section presents the details of the Privacy Model.

5.2.1 Privacy Infrastructure

We have discussed in Chapter 2 that privacy is becoming an important concern in grid and web environments where a large number of users provide their private/public information to service providers to gain access to their services/resources. The privacy model implemented by service providers must have the ability whereby users have control to check the misuse of their data as well as control over how their information can be collected, stored, used and shared. The privacy model should also allow service providers to express, evaluate and enforce privacy policies on their services/resources to protect them. A Privacy Model should address a wider notion of privacy and must be able to enforce privacy requirements of both the service providers and service requesters. It should also be integrated with the authorization framework to provide privacy based access to services/resources. To achieve this, we propose grid domains to implement privacy infrastructure as shown in Figure 5.4.

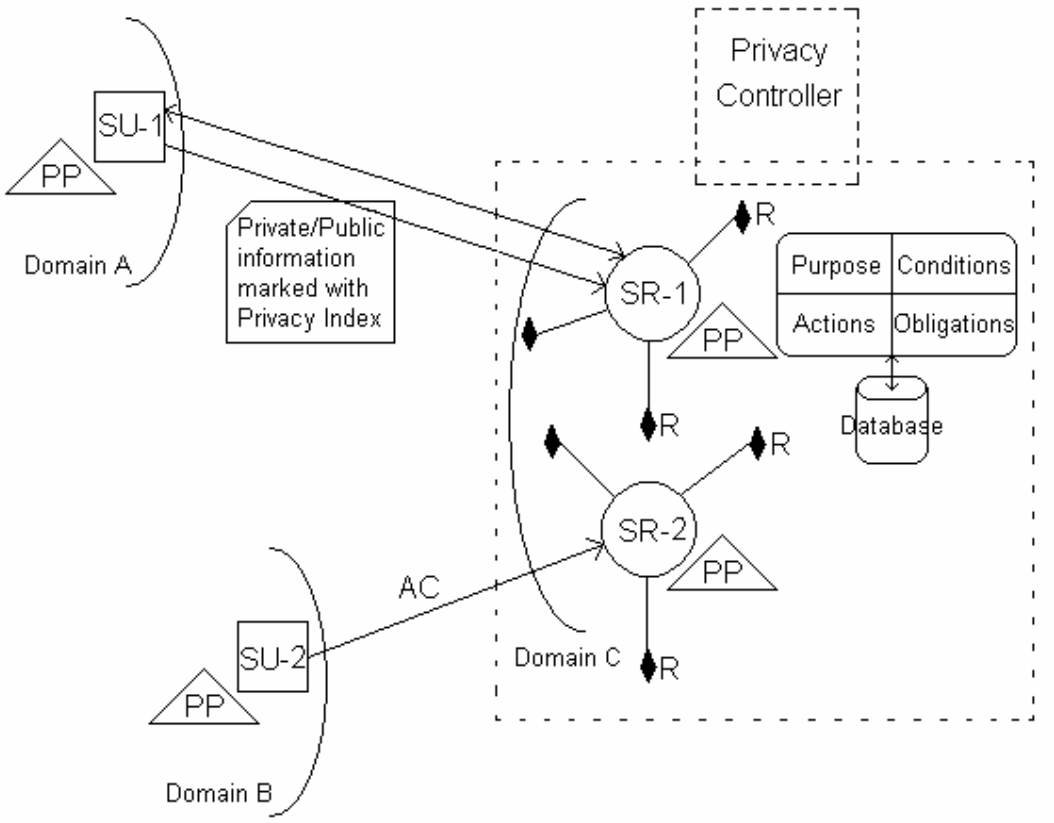

Figure 5.4: Schematic representing grid environment consisting of three domains showing how privacy requirements among subjects and services of different domains are handled.

Figure 5.4 represent a Grid Environment consisting of three Domains: Domain A, Domain B and Domain C. To understand how privacy requirements among service requesters and providers are handled, consider that SR-1 is a Service that asks for SU-1's private information. Further consider that this information is required by SR-1 to provide some other services to SU-1. Now before sending his private information to SR-1, SU-1 negotiates with SR-1 about his privacy requirements and establishes a privacy relationship. During negotiation, both agree on the information to be exchanged, the purpose for which information is required / can be used, the conditions under which information can be used, the duration for which information can be used and the parties with which information can be shared *etc.* SU-1 also marks his private information / data with PI (Privacy Index) while sending it to SR-1. All these privacy attributes *i.e.* private information, privacy index, purpose, conditions, duration, actions and obligations *etc.* are stored in database in the form of privacy policies expressed in simple XML and XACML. This interaction establishes a privacy relationship between SU-1 of Domain A and SR-1

of Domain C. In Figure 5.4, the direct association of Privacy Policies (PPs) with subjects and services is shown but actually these policies are associated with services/resources through Service Policy (SrP) because PP is a part of SrP.

A Privacy relationship has been represented as PR = (SU_a, DO_a, [R], [PU], [ACT], [CO], [OB], [SU_b], DO_b) *i.e.* Subject SU_a of Domain DO_a authorizes the set of Subjects [SU_b] of Domain DO_b to perform any of the actions specified in Action list [ACT] on set of Resources [R] for any of the purpose specified in Purposes set [PU] only if conditions specified in Conditions set [CO] are satisfied and obligations [OB] are performed. Notation "[x]" indicates the set of objects of type x. To specify more fine grained privacy relationship, PR can also be represented as (SU_a, DO_a, R, PU, ACT, CO, OB, SU_b, DO_b) i.e Subject SU_a of Domain DO_a authorizes Subject SU_b of Domain DO_b to perform Action ACT on Resource R for Purposes PU only if condition CO is satisfied and obligation OB is performed. Privacy relationships are used by authorization handler to provide privacy based access to services/resources.

The privacy relationships are kept by Domain C in such a way that later SU-1 can control and check their misuse. This is done through Privacy Controller (PC). PC is the way through which service requesters can be sure that their private information cannot be used inadequately by service providers. Whenever a privacy relationship is established between a subject and a service provider, a copy of this privacy relationship is sent to PC so that later PC can check (against this privacy relationship) whether service providers are respecting the privacy relationship established with the original subject or not.

Now consider that SU-1's private information is required by subject of some other domain, say SU-2 of Domain B. Further consider that SR-2 is a service of Domain C that provides access to SU-1's private information. Now accesses of SR-2 to SU-2 is provided only if this information is required by SU-2 to perform his task and the purpose of his task matches with the purpose for which this information was sent by SU-1. If the purpose for which SU-2 requires SU-1's private information does not match with the purpose for which SU-1 provided this information, then access is denied. To implement this functionality, the pseudo code defined in Table 5.4 has been used. A notification regarding the use of SU-1's private information is sent to SU-1, if he has asked for it. This requirement has been implemented through the concept of obligations. The violation of any established privacy relationship is checked by Privacy Controller (PC).

Following are some of the examples illustrating how privacy relationships are represented using the notation described above.

Policy 1: I authorize marketing people of organization to use my telephone number to inform me about new offers only: PR = (I, Home, telephone-no, inform-new-offer, read, null, null, marketing-people, organization)

Policy 2: Banks authorize only owners of accounts to change their personal information any time if their age is greater than 18: PR = (Manager, Bank, personal-information, null, read/write, age>18, null, owner-account, Home)

Policy 3: ABC-Travel agency will share personal information of users with government only if ordered by court and users will be notified: PR = (Manager, ABC-Travel, personal-info-users, null, read, order-by-court, notify, officials, Government)

The privacy relationships lead to privacy policies on services. The privacy policies that describe the privacy requirements of a service have been expressed in XML and privacy based access control policies have been expressed in XACML. The skeleton of these files has already been presented in Chapter 4. Section 5.2.3 describe how privacy based access to grid services/resources is provided. Privacy Model's integration with the authorization framework is explained in Section 5.4.

5.2.2 Hidden/Secret Services and Anonymous Access

Though PC plays an important role in handling service requesters' privacy requirements, it cannot be used by service providers to address their privacy requirements. To handle service provider's privacy requirements, TTP is used.

To understand how TTP has been used to address service provider's privacy requirements, consider that a service provider wants to expose a service only to a selected set of subjects and also wants to hide the security requirements associated with that service (*i.e.* the security credentials/attributes required to access the Service). Such services are called hidden/secret services. In order to provide this feature, we propose service provider to make use of TTP. Service providers will share authentication information of subjects and security requirements of services with TTP. To access a hidden/secret service, subjects pass their normal security credentials and attributes to TTP. TTP authenticates subjects (as service providers share authentication information with TTP) and if authenticated, based on the credentials/attributes provided by subjects,

TTP generates a custom security token. In custom security token, only those credentials and attributes are included which are necessary to access the service. This security token is then sent to subjects. Only this security token can be used by subjects to see and access hidden/secret services. If the subject is not authenticated or not authorized to access hidden/secret services then no custom security token is generated. Based on the information stored in custom security token, the service provider exposes some/all of the hidden/secret services that the subject is authorized to access. As subject has no knowledge of custom security token, he cannot know the actual security requirements of hidden/secret services and the actual credentials/attributes that cause access to these services. The custom security token has only limited life time and includes only those minimum credentials and attributes of the subject that are necessary to access the hidden/secret services. This mechanism provides two important privacy features to service providers:

i) Hidden/Secret services and their security requirements are not exposed to everyone.

ii) Even the authorized subjects don't know the exact security requirements of hidden/secret services. As subjects are given custom security token, they can't deduce which subset of their credentials/attributes is embedded in the custom security token and is causing access to services.

TTP also enables anonymous access on services. As service providers share authentication information of subjects with TTP, TTP can be used to authenticate subjects and generate anonymous security tokens. Anonymous security tokens assert about authorization information of subjects but remove their identity. These anonymous security tokens are used by subjects to get anonymous access to services. As anonymous security token has no information regarding the identity of subjects but only about their authorization information, service providers can't know exactly which subject is accessing the service.

Figure 5.5 shows how anonymous access to a service is provided. Here Subject SU wants to access Service SR without disclosing his identity to service provider who provides this service in Domain B. For this, SU requests TTP to generate an anonymous security token based on his identity and access rights. As service provider shares authentication and authorization information with TTP, TTP verifies the identity of SU and generates an anonymous security token. This token has limited lifetime and asserts

about authorization of SU. Thus this token can be used by SU to access service SR without disclosing his identity to service provider.

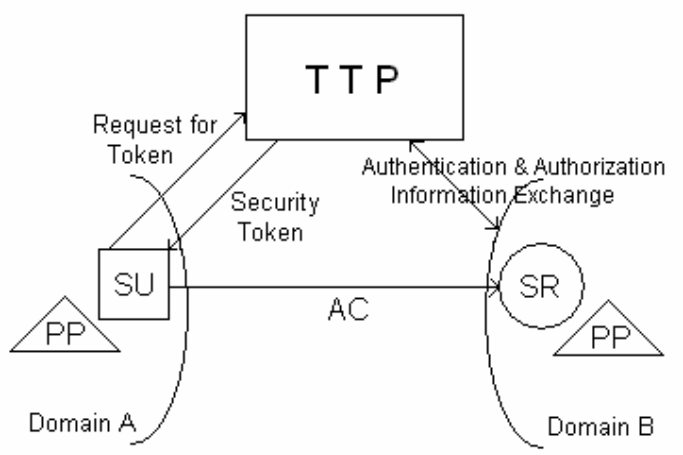

Figure 5.5: Schematic showing how anonymous access to a Service is provided

The same setup is used to access hidden/secret services also. If SU wants to access a secret service provided by Domain B then he requests a custom security token from TTP by presenting him his normal security credential. TTP verifies the identity of SU, fetches SU's attributes from Domain B, generates a custom security token and sends it to SU. Now SU can present this custom security token to Domain B to see and access all hidden/secret services that he is authorized to access.

5.2.3 Privacy based Access

The Privacy Model also supports privacy based access (or more precisely purpose based access). The established privacy relationships can be used by authorization framework to provide privacy based access to services. The following sequences of steps have been used in the model to provide privacy based access to a service/resource when a subject requests it for some purpose.

Step 1: Extract security credentials of the Subject from request and verify the identity. If Subject authentication fails then quit otherwise go to Step 2.
Step 2: Determine whether Subject conforms to Privacy Policy (PP) associated with that

Service/Resource. For this, perform the following:

i) Get the list of authorized Subjects ([SU]) that can access the requested Service/Resource.

ii) If Subject's identity does not appear in the list of authorized Subjects then quit otherwise do the following.

iii) Obtain the purpose set [PU] associated with the requested Service/Resource for which it can be accessed.

iv) Extract the purpose of access from access request.

v) If the purpose for which Subject wants to access the Service/Resource is not listed in the set of purposes ([PU]) then quit otherwise perform the following.

vi) If the access action specified in the Subject's access request is not listed in the set of actions allowed ([ACT]) on that Service/Resource for that particular purpose then quit otherwise perform the following.

vii) Obtain privacy statements and conditions associated with that Service/Resource from the policy database and, the Subject and Resource attributes from the attribute database.

viii) Check whether the conditions specified in [CO] set are satisfied. If not then deny access otherwise proceed next.

ix) Check whether the obligations specified in the [OB] set can be performed. If not, then quit otherwise perform Step 3.

Step 3: Provide access of Service/Resource to Subject.

Step 4: Perform obligations specified in the [OB] set.

Step 5: Log access actions in log tables to address auditing and accounting requirements.

Table 5.4: Sequence of steps for privacy based access to grid services/resources

The above sequence of steps guarantee that if a service is exposing subject's private information then that information can be accessed by other subjects/services only if they are authorized and the purpose for which they are accessing the information matches with the purpose for which the information was actually sent by the subject to the service provider. These steps are integrated with overall authorization framework to provide privacy based access to services/resources. As access to resources depend on many other factor also (besides privacy), authorization framework make use of other relevant components to arrive at final authorization result.

Privacy Controller can be used to make sure for subjects that service providers are providing privacy based access to private information according to the privacy relationships that they have established initially. Thus subjects can be sure that their private information can not be misused / accessed inadequately from service provider's database.

5.2.4 Services defined, implemented and exposed by Privacy Model

Privacy model has been implemented as a collection of privacy related services exposed as web services. Some of the services defined, exposed and used in the implementation are:

Service Name	**Purpose**
getAuthorizedSubjects( )	To get a list of authorized subjects associated with a service for a particular purpose for a particular action.
getPurpose( )	To get purpose associated with a service for a particular action.
getAllowedAction( )	To get allowed actions on a service for a particular purpose.
getPrivacyPolicy( )	To get privacy policy associated with a service.
getPrivateServices( )	To get a list of services that require private data.
getAnonymousServices( )	To get a list of services that can be anonymously accessed.
getSecretServices( )	To get a list of services that are hidden.
getPrivacyIndex( )	To get privacy index associated with a resource/service.
setAuthorizedSubjecst( )	To set authorized subjects for a particular service for a particular purpose and action.
setPurpose( )	To set purpose associated with a service for a particular action.
setAllowedActions( )	To set allowed actions on a service for a particular

	purpose.
setPrivacyPolicy( )	To set privacy policy associated with a service.
setPrivacyIndex( )	To set privacy index associated with a service /resource.
isPrivate( )	To find whether a service requires private data.
isAnonymous( )	To find whether a service supports anonymous access.
setPrivacyType( )	To set the privacy type of a service (hidden / anonymous / private *etc*.)
removeAuthorizedSubjects( )	To remove subjects associate with a service for a particular action or purpose.
removePurpose( )	To remove purpose associated with a service for a particular action.
removeAllowedActions( )	To remove allowed actions associated with a service for a particular purpose.
removePrivacyPolicy( )	To remove privacy policy associated with a service.

Table 5.5: Services defined, implemented and exposed by privacy model

These services have been implemented in .NET environment as web services. Most of the services listed in Table 5.5 are overloaded and represent a subset of the total services exposed by the privacy model.

5.3 Trust Model

A Trust Model can be defined as a system that allows service requesters and service providers to assess trustworthiness of each other as well as state, evaluate and enforce trust relationships among them. It is a detailed description of all aspects of a system that relate to trust. A trust based authorization system grants specific type of access to specific requesters based on their authentication, trust status, what services/resources they are accessing, current state of the system and their conformance to established trust and other security policies. Trust Model can also be integrated with the authorization framework to

provide trust based access to services. Following sections present the details of the implemented Trust Model.

5.3.1 Trust Relationships

Determining trust relationship is essential not only while accessing resources/services but also while enabling delegation. A subject may decide to delegate rights to target service/subject only if trust with that service/subject is greater than a threshold value. A subject may also decide to send his/her private data/information to target service/service provider/domain based on the value of trust it has with the target service/service provider/domain. Trust can take a complex form in distributed environment. The trust status of a service/resource from a different domain is hard to determine. Subjects generally have no idea whether the service is compromised or is malicious [103]. There may occur different trust relationships among members of different domains. *e.g.* Subjects of Domain A trust Domain B and not Domain C, or, Subjects trust Domain B for Service 1 but not for Service 2, *etc.* The main problem is how to establish and manage trust relationships between subjects and services of different domains and how to determine the trustworthiness of target service/service provider/domain. Domains need a trust enabled security infrastructure to handle trust related requirements and to provide trust based access to services/resources. Establishment of trust may be a one time activity per session or it may be dynamic [43] (requiring the establishment of trust for each request). The trust of an entity with other entity is not a fixed value but can change dynamically depending on the past and present behavior of the entity and context in the environment.

Trust should be established from the viewpoint of both the parties (Service Requesters and Service Providers). Requester's trust with the service provider may be different from the service provider's trust with the requester. This situation is modeled using two different types of trust: code trust and execution trust, as defined in [103]. Execution trust exists from subject's side to service provider's side that service provider will correctly and faithfully allocate resources for the efficient execution of a job with respect to established trust and other security policies. Code trust exist from service provider's side to subject's side that subject will generate a legitimate request consisting of virus free code and will not produce malicious results and does not temper other

results/information/code present at service provider's end. In addition to execution and code trust, following terminology regarding trust has also been used in the security policy framework:

Direct Trust: is the trust that a subject holds on a service /service provider/ domain without any intermediate service/service provider/domain or other entity.

Full Trust: A subject is said to have full trust on a service provider/domain if it trust on all the services provided by that service provider/domain.

Partial Trust: A subject is said to have partial trust on a service provider/domain if it trusts on some of the services provided by that service provider/domain.

Recommended Trust: is the trust of one entity on second entity that is recommended by other entities.

Privacy Trust: is the trust of an entity on the privacy features provided by other entity.

Authentication Trust: is the trust of an entity on the authenticity of an identity certificate signed by a certificate authority.

We propose trust relationship to be represented as: TR = (SU_1, DO_1, TT, SR_1, DO_2, c, t, tv) *i.e.* Subject SU_1 of Domain DO_1 trust Service SR_1 of Domain DO_2 with trust value tv with respect to trust type TT at time t for context c. TT represent the type of trust which can be Authentication Trust, Privacy Trust, Authorization Trust, Direct Trust, Recommendation Trust *etc.* We also propose Trust Model to be a system represented as TS = (GES, TR, OP), where GES refers to different entities in a grid environment among which trust relationship exist. These entities can be subjects, services, service providers or domains *etc.* TR is the type of trust (as discussed above) that can exist among these entities and OP is the set of operators used to determine trustworthiness of one entity on another. Next section describes the approach used in the implemented model to determine the trustworthiness of target entity.

5.3.2 Trust Evaluation

If a subject from a particular domain wants to access a service from another domain, then trust value *tv* can be calculated as:

$$tv = \alpha * dt + \beta * rt \qquad \text{(1)}$$

where *dt* is Direct Trust and *rt* is Recommended Trust. α and β are the weights assigned to direct and recommended trust. In general, direct trust is assigned more weightage over recommended trust. α and β are chosen such that $\alpha + \beta = 1$. Let *s* and *f* denote the success and failure evidences experienced by the subject about the service. Then direct trust can be calculated as:

$$dt = s / (s + f) \quad \text{.....} \quad (2)$$

and recommended trust can be calculated as:

*rt = Average of (subject's trust on R1 * R1's trust on service,*
*subject's trust on R2 * R2's trust on service,*
.....
.....
*subject's trust on Rn * Rn's trust on service)* ..... (3)

where *R1, R2, …, Rn* are recommenders of the subject. Recommenders are trusted by subjects to make recommendations about trustworthiness of other services. We are making use of these equations to determine the trustworthiness of one entity on other entity. In case of full trust, *f* can be set to 0 and in case of partial trust, *s* and *f* take different values depending on the level of trust. In case of no trust, *s* can be set to 0. A history of these values is maintained by every domain. To calculate *s* and *f*, last *n* interactions are considered where *n* can be any number depending on the requirements. A large value of *n* models accurate behavior whereas a small value of *n* models current behavior of the entity. To update trust, changes are made to *s* and *f*. Based on the values of *dt* and *rt*, trustworthiness of an entity can be categorized into different levels as shown in Table 5.6.

Level	***dt***	***rt***	***tv* (with α=β=0.5)**	**Meaning**
L1	>0.5	>0.5	>0.5 and <=1	Trust (White Zone)
L2	>=0.5	<=0.5	<=0.75	Can be trusted with some risk (Gray Zone)
L3	<=0.5	>=0.5	<=0.75	Can be trusted but greater risk (Gray Zone)
L4	<0.5	<0.5	<0.50	Don't Trust (Black Zone)

Table 5.6: Levels of trust with α = 0.5 and β = 0.5

Table 5.6 shows four levels of trust ranging from reasonable trust (White Zone) to no trust (Black Zone) depending upon the *dt* and *rt* values. *dt* and *rt* can take a maximum value of 1. *tv* can also take a maximum value of 1. The more the value of *tv* is, the higher is the trust. A *dt* and *rt* value greater than 0.5 means that both the subject and the recommender have more success evidences than failure. This represents a White Zone and target can be trusted with a satisfactory level of trust. If a subject experiences more success evidences and recommender experiences failure evidences, or vice versa, then the trustworthiness of target is doubtful. This represents Gray Zone. At this point, a decision should be made whether subject wants to give more weightage to recommender trust (*rt*) or his direct trust (*dt*) on service. In case where $\alpha = \beta = 0.5$ *i.e.* both, the direct and recommender trust are given equal weightage, these two levels are the same. Generally, *dt* value is given more weightage over *rt* value. So if a subject experiences more success evidences and recommender experiences failure evidences, then the trustworthiness of target is less doubtful compared to subject being experiencing failure evidences and recommender experiencing more success evidences. If the subject and recommender, both have more failure evidences than success evidences, then the target is definitely not to be trusted. This represents a Black Zone. Two important data structures that have been maintained for the calculation of trust value (*tv*) are:

i) Trust Table

ii) Recommender Table.

Each Domain maintains a copy of these tables. Following is the basic layout of these tables:

Trust (sourceDomain, sourceServiceProvider, sourceSubject, targetDomain, targetServiceProvider, targetSubject, success, failure, threshold, context, time)

Recommender (recommenderDomain, recommenderServiceProvider, recommenderSubject, context, time)

Using above equations and data structures, trust of a subject on subjects/services of other domains can be calculated. For this, we are making use of the pseudo codes presented in Table 5.7, Table 5.8 and Table 5.9 that make use of these equations and data structures.

calculateTrust (S-DO, S-SP, S-S, T-DO, T-SP, T-S, c)

S-DO: Source Domain

S-SP: Source Service Provider

S-S: Source Subject

T-DO: Target Domain

T-SP: Target Service Provider

T-S: Target Subject

c: context

tv: trust value

dt: direct trust

rt: recommended trust

α: direct trust weightage

β: recommended trust weightage

i) Set tv = 0, dt = 0, rt = 0, α = 0.6, β=0.4.

ii) Set dt= calDirectTrust (S-DO, S-SP, S-S, T-DO, T-SP, T-S, c)

iii) Set rt = calRecTrust (S-DO, S-SP, S-S, T-DO, T-SP, T-S, c)

iv) Set tv= α * dt+ β * rt

v) Return tv

Table 5.7: Pseudo code to calculate trust value

calDirectTrust (S-DO, S-SP, S-S, T-DO, T-SP, T-S, c)

S-DO: Source Domain

S-SP: Source Service Provider

S-S: Source Subject

T-DO: Target Domain

T-SP: Target Service Provider

T-S: Target Subject

c: context

dt: direct trust

i) Set dt = 0.

ii) Select success, failure from Trust where sourceDomain=S-DO and sourceServiceProvider=S-SP and sourceSubject=S-S and targetDomain=T-DO and targetServiceProvider=T-SP and targetSubject=T-S and context=c

iii) Set dt=success / (success + failure)

iv) Return dt

Table 5.8: Pseudo code to calculate direct trust

calRecTrust(S-DO, S-SP, S-S, T-DO, T-SP, T-S, c)

S-DO: Source Domain

S-SP: Source Service Provider

S-S: Source Subject

T-DO: Target Domain

T-SP: Target Service Provider

T-S: target Subject

c: context

nr: number of recommenders

i: loop variable

str: source's trust on recommender

rtt: recommender's trust on target

CR-D: current recommender Domain

CR-SP: current recommender Service Provider

CR-S: current recommender Subject

rt: recommended trust

i) Set rt = 0

ii) Set nr = Select count (*) from Recommender

iii) for (i =1; i<nr; i++)

a) Set CR-D= Select recommenderDomain[i] from Recommender

b) Set CR-SP = Select recommenderServiceProvider[i] from Recommender

b) Set CR-S= Select recommenderSubject[i] from Recommender

c) Select success, failure from Trust where sourceDomain=S-DO and sourceServiceProvider=S-SP and sourceSubject=S-S and targetDomain=CR-D and targetServiceProvider=CR-SP and targetSubject=CR-S and context=c

d) Set str = success / (success + failure)

e) Set rtt = getDirTrust (CR-D, CR-SP, CR-S, T-DO, T-SP, T-S)

f) rt = rt + str*rtt

iv) Set rt = rt/nr

v) Return rt

Table 5.9: Pseudo code to calculate recommended trust

Table 5.7 represents pseudo code to calculate trust value. As shown, the direct trust, recommender trust and trust value are initially set to 0. Direct trust weightage (α) and recommender trust weightage (β) are set to 0.6 and 0.4 respectively as generally direct trust is given more weightage over recommender trust. Different values of α and β can also be taken to implement different weighting strategy. Then the direct trust of the source subject on the target service is calculated using the calDirectTrust( ) algorithm, which is shown in Table 5.8. To calculate direct trust, the success and failure evidences experienced by the source subject with target service are fetched from the Trust table and *dt* is calculated using Equation (2). This is shown in step iii of calDirectTrust( ). To calculate recommended trust, calRecTrust( ) algorithm is used which is shown in Table 5.9. After determining *dt* and *rt* values, trust value *tv* is calculated using Equation (1). This is shown in step iv of calculateTrust( ) algorithm. To calculate recommended trust, first, the trust of the source subject on recommender is calculated and then the recommender's trust on target is called from the recommender. This is shown in steps iii-d and iii-e of calRecTrust( ) algorithm. The process is repeated for all the recommenders in the Recommender table. After getting the recommendation from all the recommenders, average value of *rt* is calculated using Equation (3). This is shown in steps iii-f and iv of calRecTrust( ) algorithm.

Using these algorithms, we can set up trust enabled authorization framework that determine the trustworthiness of one entity on another and provide access to target service based on the trust value. A threshold value of *tv* can be set depending on the level of trust required. After authenticating subject, we can calculate *tv* value and check whether *tv* is greater than or less that threshold value. If $tv \geq$ threshold value then access to service/resource is provided otherwise access is denied. After each access, following updation are made to *s* and *f* values:

$s = s + 1$, if source is satisfied by the service/resource access provided by the target, otherwise no change.

$f = f + 1$, if source is not satisfied by the service/resource access provided by the target, otherwise no change.

5.3.3 Trust Manager Service

In order to establish and evaluate trust among different entities in a grid environment, we propose every domain to implement trust model integrated with the authorization framework. Besides handling trust related security requirements and issues, trust model will also provide trust based access to services/resources. Figure 5.6 shows a high level view of the trust model.

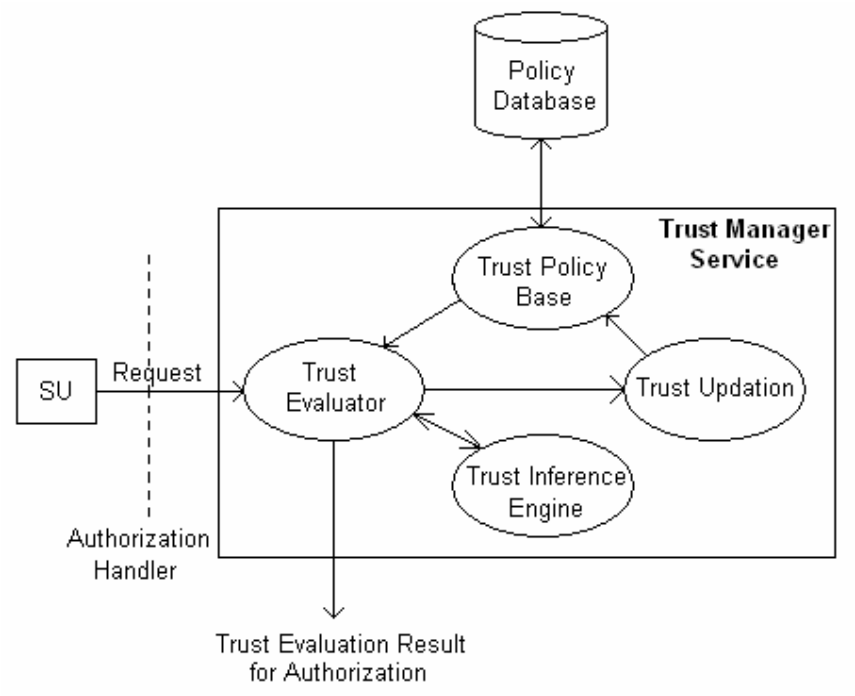

Figure 5.6: Trust Model architecture for trust based access

The Trust Model has been implemented as a Trust Manager Service. As shown in Figure 5.6, access request coming from subject, through authorization handler (explained in Section 5.4), is first intercepted by Trust Evaluator component. Trust Evaluator is responsible for evaluating trust based on established trust relationships and providing trust evaluation result to authorization handler. To evaluate trust, Trust Evaluator makes use of Trust Inference Engine. Trust Inference Engine calculates *tv* value and passes it to trust evaluator for use in preparing final result. Trust Evaluator is also responsible for updation of trust among entities. Trust is updated with every interaction that takes place between a subject and a service. Trust Policy Base fetches established trust relationships and policies from policy database and provides these to trust evaluator for evaluating trust

and preparing final result. Trust Evaluator then passes this result to authorization handler. The trust model is capable of capturing different types of trust and provides mechanism for trust evaluation, recommendations and updation of trust. Trust related access control policies have been expressed in XACML [41]. The format of XACML policy has already been discussed in Section 4.3. Trust model's integration with the authorization framework is described in Section 5.4.

5.3.4 Services defined, implemented and exposed by Trust Model

Trust model has been implemented as a collection of trust related services exposed as web services. Some of the services defined, exposed and used in the implementation are:

Service Name	**Purpose**
getTrustOnDomain( )	To get trust on a domain.
getTrustOnServiceProvider( )	To get trust on a particular service provider.
getTrustOnService( )	To get trust on a service.
getDirectTrust( )	To get direct trust on a service.
getRecommendedTrust( )	To get recommended trust on a service.
getRecommender( )	To get a list of recommenders for a service/service provider/domain.
getTrustPolicy( )	To get trust policy associated with a service.
getTrustThreshold( )	To get trust threshold associated with a service.
getTrustedDomains( )	To get a list of trusted domains.
getTrustedServiceProviders( )	To get a list of trusted service providers.
getTrustedServices( )	To get a list of trusted services.
getSuccessEvidences( )	To get success evidences experienced with a service for a particular context.
getFailureEvidences( )	To get failure evidences experienced with a service for a particular context.
getTrustContext( )	To get context associated with a trust relationship.
addRecommender( )	To add recommenders for a particular service/service provider/domain.

setTrustTheshold( )	To set trust threshold associated with a particular service.
setTrustPolicy( )	To set trust policy associated with a service.
removeRecommender( )	To remove recommenders for a particular service/service provider/domain.
removeTrustPolicy( )	To remove trust policy associated with a service.
updateTrustValues( )	To update trust values (success / failure evidences experienced) for particular service.

Table 5.10: Services defined, implemented and exposed by trust model

Like other services, these services have also been implemented as web services in .NET environment. Most of the services listed here are overloaded and represent a subset of the total services exposed by the trust model.

5.4 Policy based Authorization Model

As discussed in Chapter 2, authorization system can be defined as a system that grants specific type of access to specific requesters based on their authentication, what services/resources they are accessing, current state of the system and their conformation to established authentication, privacy, trust and other security policies. It is a detailed description of all aspects of a system dealing with access of services/resources by requesters. Authorization system should be scalable and must be able to enforce different types of security policies. Following sections present the details of the implemented framework.

5.4.1 Authorization System

In grids, the number of subjects and services/resources are very large. The services/resources also have different authentication, privacy, trust and authorization requirements. Subjects have different roles/privileges in different domains, which results in their different authorization status in different domains. If the number of requesters requesting the services of a target domain is very large then it will be very difficult for the

target domain to maintain access rights information of all of its requesters. To address this problem, we propose target domain to store access rights information of all of the users/requesters from a source domain as a whole (*i.e.* the access rights that target domain grants to source domain irrespective of a particular user/requester) and not of the individual users/requesters from the source domain. The source domains will themselves maintain access rights information of their respective subjects. This approach will make the authorization system more scalable. This approach has been implemented in the framework using Filter-In and Filter-Out components. Figure 5.7 illustrates this concept.

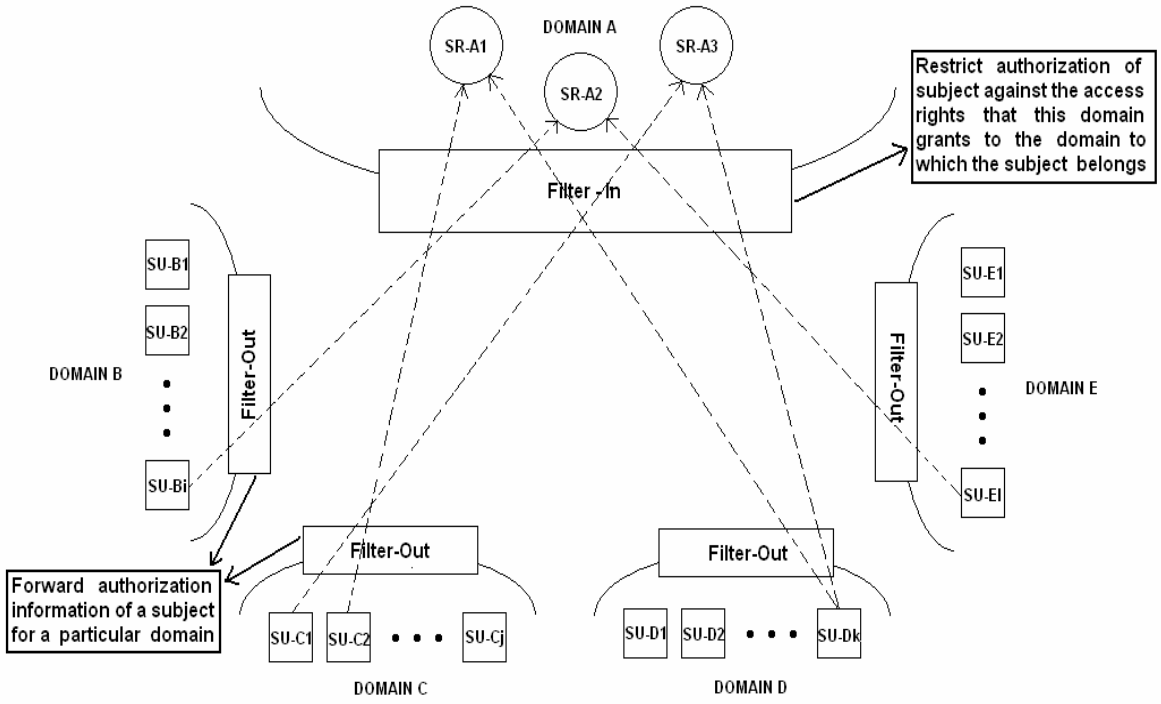

Figure 5.7: Schematic showing the use of Filter-In and Filter-Out components

As shown in Figure 5.7, there are five domains: A, B, C, D and E. The domains B, C, D and E have i, j, k and l number of subjects respectively. These subjects access services provided by domain A. Each of these subjects has different roles/privileges in domain A. If the number of subjects in domain B, C, D and E is very large then it will be very difficult for domain A to maintain access rights information of all of these subjects from different domains. As a solution to this problem, we propose domain A to store access

rights information of all the subjects of a particular domain as a whole and not of individual subjects of individual domain. This means that domain A will maintain

i) access rights information that it grants to domain B irrespective of particular subjects of Domain B

ii) access rights information that it grants to domain C irrespective of particular subjects of Domain C

iii) access rights information that it grants to domain D irrespective of particular subjects of Domain D

iv) access rights information that it grants to domain E irrespective of particular subjects of Domain E

The exact access rights information of a subject for services of Domain-A will be maintained by the domain to which the subject belongs.

To achieve this functionality, we are making use of Filter components. An example usage of Filter components has been explained in Chapter 3. There are two types of Filters: Filter-In and Filter-Out. Through Filter-Out component, the subject leaves his source domain with access rights that his source domain grants to him. It attenuates/filters the access rights of a subject for a particular target domain. Through Filter-In component, the subject enters the target domain with access rights that the target domain grants to the source domain. In other words, the subject gets the intersection of the rights that his source domain grants to him and the rights that target domain grants to the source domain. In this particular example, Filter-Out components of Domains B, C, D and E will maintain access rights information of their respective subjects and Filter-In component of Domain A will maintain access rights information that Domain-A grants to Domain B, Domain C, Domain D and Domain E irrespective of their subjects. This makes the authorization system more scalable as a single domain is not maintaining access rights information of all the subjects instead subject wide access rights information is maintained by the source domain to which subject belongs and the domain wide access rights information is maintained by the target domain.

Filter-In and Filter-Out components have been implemented as services which make assertions about access rights information of subjects and get called when subjects issue access request on a service.

5.4.2 Policy Based Access System

As described in Chapter 3, the access control policies of a Domain, Service Provider and Service have been expressed in XACML. The powerful XACML policy language model allow us to specify much fine grained access control policies on services/resources using its different elements, which is not possible through other mechanisms. To represent context of a particular access request, the addition of "context" element in XACML policy language model is proposed. Figure 4.5 illustrates the placement of proposed "context" tag in XACML policy language model. In the implemented framework, XACML and its extension capabilities are allowing us to specify fine grained and context based access control policies on services/resources.

After determining the authorization information of subjects, their conformance to domain policy (DP), service provider policy (SPP), service policy (SrP) and other applicable policies is checked. If subject conforms to all these policies, then access is granted otherwise access is denied. Following sequence of steps represent a high level description of the process of providing access to a requested service/resource.

Step 1: Extract security credentials of subject from request and verify the identity. If subject authentication fails then quit otherwise go to Step 2.

Step 2: Determine authorization information of subject using Filter components. If subject is not authorized to access the service then quit otherwise go to Step 3.

Step 3: To determine whether subject conforms to all applicable security policies, extract the Domain Policy (DP), Service Provider Policy (SSP), Service Policy (SrP) and the policies for which target matches with access request attributes from the policy database, and perform the following for each of the extracted policies:

i) Call authentication policy handler to determine whether subject conforms to authentication policies. If not then quit otherwise proceed next.

ii) Call privacy policy handler to determine whether subject conforms to privacy policies. If not then quit otherwise proceed next.

iii) Call trust policy handler to determine whether subject conforms to trust policies. If not then quit otherwise proceed next.

iv) Determine whether subject conforms to other access control policies. If not then quit otherwise proceed next.
Step 4: Check whether the obligations applicable for this access can be performed. If not, then quit otherwise perform Step 5.
Step 5: Provide access to requested service/resource.
Step 6: Perform applicable obligations.
Step 7: Log access actions in log tables to address auditing and accounting requirements.

Table 5.11: Sequence of steps for policy based access to grid services/resources

To achieve the functionality described above, we are proposing and making use of the XACML based authorization model as shown in Figure 5.8. As DP, SPP and SrP can have authentication, privacy and trust related policies also, the authentication, privacy and trust handler have also been integrated in the framework to evaluate their respective policies. The authentication handler functionality has been implemented in PDP itself so it is not shown in Figure 5.8 but privacy and trust handlers are explicitly shown. The major components of the model are PEP, PDP and PIP. PEP (Policy Enforcement Point) performs access control by making decision requests and enforcing authorization decisions. PDP (Policy Decision Point) evaluates applicable policy and renders an authorization decision. PIP (Policy Information Point) acts as a source of attribute values.

The authenticated and filtered request of Subject SU is first intercepted by PEP. PEP constructs an authorization decision query and passes it to authorization handler. The result of this query determines whether access to service/resource is granted or denied. The authorization decision query has details about the identity of the subject, the service requested, the purpose of access *etc.* Authorization handler passes this information to PDP. PDP is responsible for evaluating policies. The Policies are retrieved by PDP from PRP (Policy Retrieval Point). If the policy information is not available at PRP, it may be retrieved from Policy Store. The Policy Store is capable of importing/exporting policies. The policies are written by administrator using PAP (Policy Administration Point).

PDP is proposed to be implemented as a combination of other PDPs (PDP-1, PDP-2 … PDP-n). These components (PDP-1, PDP-2, … PDP-n) will implement policy decision functionality specific to a particular technology/mechanism. As the rules to express, store and interpret policies can be different in different mechanisms, separate PDPs are required to determine policy decision specific to a particular mechanism. *e.g.* there can be

one PDP for access control lists, one for role based access control and another for SAML and XACML based access *etc*. As PDP is responsible for evaluating policy, it may require subject, resource, environment and other attributes for the evaluation of that policy. The attribute information is requested by PDP from authorization handler which in turn requests this information from PIP.

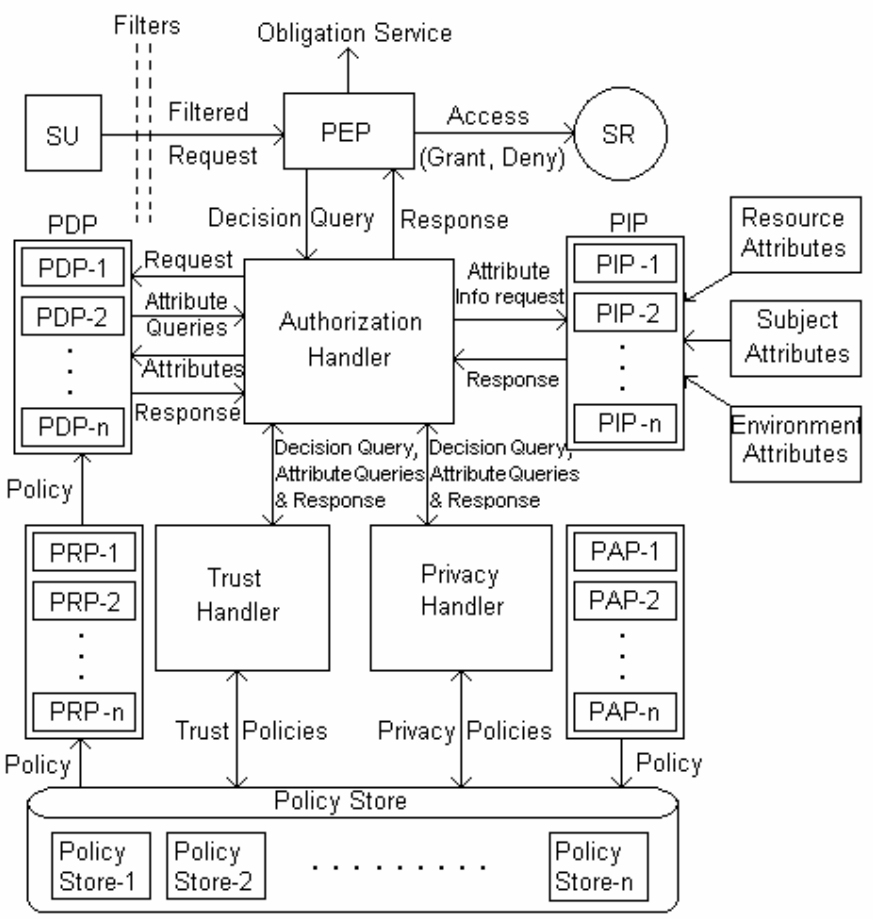

Figure 5.8: Integrated Policy Based Authorization Model

PIP (Policy Information Point) is used by authorization handler to retrieve subject, resource and environment related attributes. Like PDP, PIP is also proposed to be implemented as a combination of other PIPs (PIP-1, PIP-2, … PIP-n). As subject, resource, environment and other attributes might be stored in different formats at different places, separate mechanisms are required to fetch, understand and supply those attributes to authorization handler. PIP passes attribute information to authorization handler which in turn passes this information to PDP. Now PDP evaluate the policy and prepare the

result which is then passed to authorization handler. Trust handler and privacy handler provide trust and privacy based access information to authorization handler by evaluating trust and privacy relationships and policies. The authentication handler functionality has been implemented in PDP itself. After getting all the information from PDP, trust handler and privacy handler, authorization handler prepares a final result and passes it to PEP. Based on the result, PEP then grants/denies access to the requested service/resource. Obligation service, if any, is also executed by PEP.

5.4.3 High Level View of Implementation

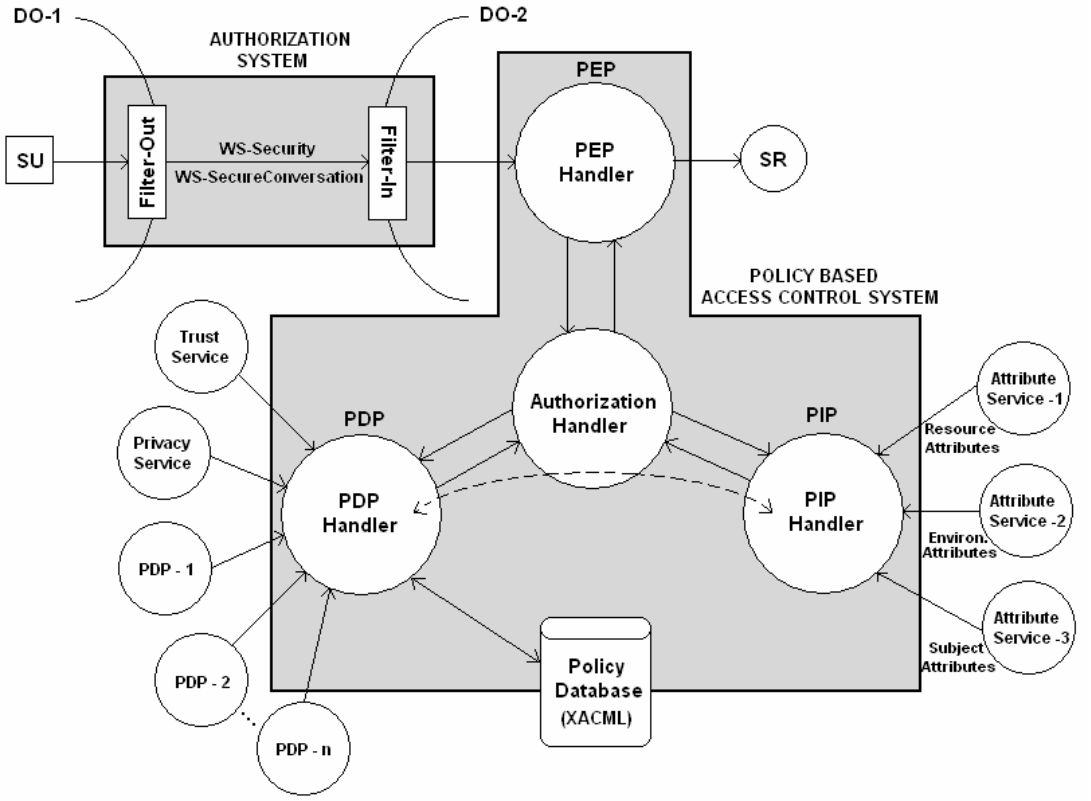

Figure 5.9: Schematic showing high level view of the implementation

Figure 5.9 shows a high level view of the implementation. As shown in Figure 5.9, subject's access request for Service SR first passes through Filter-Out component at the source domain and then through Filter-In component at the target domain. During this passage, subject's access rights are filtered for the target domain. At the target domain, PEP prepares authorization decision query and passes it to authorization handler. PEP,

PDP and PIP have been implemented as web services. Privacy and trust handlers have also been implemented as trust and privacy services. PDP handler makes use of trust and privacy services to get trust and privacy related access control information. It also gets information from other PDP implementations (PDP-1, PDP-2 … PDP-n). PDP makes use of policy database to fetch policies applicable to a particular access request. These policies are evaluated and evaluation result is prepared. PIP obtains attributes related to different aspects using attribute services and passes these to authorization handler. Authorization handler then passes these attributes to PDP for policy evaluation. Thus attributes required for evaluation come to PDP from PIP through authorization handler. This interaction is shown as dotted curved line between PDP handler and PIP handler in Figure 5.9. PDP prepares evaluation result and passes it to authorization handler. After getting all the information, authorization handler prepares final result which is then passed to PEP. Based on the response received from authorization handler, PEP either grants or denies access to requested service/resource. During all these steps the relevant information is stored in log tables also to address auditing and accounting requirements.

5.4.4 Services defined, implemented and exposed by Integrated Policy Based Authorization Model

Integrated Policy-based authorization model has been implemented as a collection of authorization and policy related security services exposed as web services. Some of the services defined, exposed and used in the implementation are:

Service Name	**Purpose**
getDomainPolicy( )	To get policy associated with a domain.
getServiceProviderPolicy ( )	To get policy associated with a service provider.
getServicePolicy( )	To get policy associated with a service.
authenticate( )	To authenticate a subject. This service is overloaded.
evaluateAuthenticationPolicy( )	To evaluate authentication policy associated with a service.
evaluatePrivacyPolicy( )	To evaluate privacy policy associated with a

	service.
evaluateTrustPolicy( )	To evaluate trust policy associated with a service.
evaluateAuthorizationPolicy( )	To evaluate authorization policy associated with a service.
evaluateSecurityPolicy( )	To evaluate security policy associated with a service.
getSourceDomain( )	To get source domain from request.
getSourceServiceProvider( )	To get source service provider from request.
getSourceSubject( )	To get source subject from request.
getAccessContext( )	To get context associated with the access request.
getSubjectAttributes( )	To get attributes associated with a subject.
getResourceAttributes( )	To get attributes associated with a service/resource.
getAccessibleDomains( )	To get list of accessible domains.
getBlackListedDomains( )	To get list of black listed domains.
getXACMLPolicy( )	To retrieve “policy” element from XACML policy.
getXACMLRule( )	To retrieve “rule” element from XACML policy.
getXACMLTarget( )	To retrieve “target” element from XACML policy.
setServicePolicy( )	To set policy associated with a service.
setServiceProviderPolicy ( )	To set policy associated with a service provider.
setDomainPolicy( )	To set policy associated with a domain.

Table 5.12: Services defined, implemented and exposed by integrated policy based authorization model

Most of the services listed above are overloaded and represent a subset of the total services exposed by integrated policy based authorization model.

5.5 Distinguished Features of Models

This section lists important features of the implemented models that make them different from other efforts in the relevant areas as described in the Chapter 2. Following paragraphs describe the important characteristics of each model.

The authentication model provides support for single sign-on and delegation features using proxy certificates and a credential management service to store, retrieve and update multiple user credentials. Following are the important characteristics of the authentication model.

- It provides credential repository and a service to manage multiple user credentials.
- It is based on Web Services Security specifications like WS-Security, WS-SecureConversation *etc.*
- It supports the storage of multiple types of tokens/credentials. (e.g. X.509, Kerberos, Custom Security Token *etc.)*
- It can be used for grid as well as web services.
- CMS is distributed over different domains, therefore removes the drawbacks of central storage.
- It provides support for single sign-on and delegation through proxy certificates.
- It provides a mechanism to express, evaluate and enforce authentication policies.

The privacy model provides support for anonymous access, hidden service access and access to private information based on conformance to privacy policies. Following are the characteristics of privacy model.

- It supports purpose based access to private information/resources.
- It supports hidden service access through custom security tokens.
- It supports anonymous access through the concept of anonymous security tokens.
- It provides mechanisms to handle privacy requirements of both, the service requesters as well as service providers.
- It provides mechanisms to express, evaluate and enforce privacy policies between service providers and requesters.
- It describes the integration of privacy based access with authorization framework.
- It uses XACML to express access control privacy policies which is OASIS standard.

The trust model provides support for calculating direct as well as recommended trust to determine trustworthiness of target service for enabling trust based access to grid services. Following are the important characteristics of the trust model.

- It can deal with identity as well as behavior trust.
- It provides support to calculate direct as well as recommended trust.
- It provides mechanisms to express, evaluate, and enforce trust policies.
- It describes the integration of trust based access with authorization framework.
- It also supports updation and management of trust relationships.

Integrated policy based authorization model provides policy based access to grid services and integrates with privacy and trust models by implementing trust and privacy based access. It provides support to express, evaluate and enforce different types of security policies (authentication, privacy, trust and authorization policies). It enables us to treat and specify the policy involving combination of authentication, privacy, trust and authorization related aspects as a single unit. The framework is flexible, scalable, supports fine grained access to services/resources and is able to express and enforce VO wide and other access control security policies. Following are the important characteristics of the integrated policy based authorization model.

- It supports fine grained and context based access to grid services/resources.
- Policy expression is platform independent.
- It is able to express and enforce VO wide and service wide access control policies.
- It introduces Filter components which make it flexible and scalable.
- It extends basic authorization mechanism to include trust and privacy based access to grid services.
- It supports multiple security policies and provides facilities to integrate different authorization mechanisms.

All these features make the implemented framework distinguished from other efforts. Next Chapter describes the implementation results and performance analysis of authentication, privacy, trust and authorization policies. The framework has been evaluated by implementing various security related scenarios and through different implementations that involve enforcement of different types of access control policies. The implementation has been done in .NET environment with the support of WSE 3.0 toolkit.