SUMMARY

1. Introduction

Large-scale distributed computing environments, or “computational grids" as they are sometimes termed couple computers, storage systems, and other devices to enable distributed applications.

Grid applications are distinguished from traditional client-server applications by their simultaneous use of large numbers of resources, dynamic resource requirements, use of resources from multiple administrative domains.

There are three major problems addressed by grid computing:

Computer-centric problems. The user needs computation resources, as many as possible. These computer-centric applications can benefit from the Grid to combine large computational resources in order to tackle problems that cannot be solved on a single system.

Data-centric problems - also called data-intensive problems - are the primary problems addressed by grid computing at present. (systems used to collect, store and analyze scientific data (e.g particle and astrophysics experiments), monitoring systems).

Community-centric problems, also referred to as collaborative applications, are concerned primarily with enabling and enhancing human-to-human interactions, attempting to bring people or communities together for collaborations of various types. (examples: interactive video presentation and conferencing from many sites simultaneously).

While scalability, performance and heterogeneity are desirable goals for any distributed system, the characteristics of computational grids lead to security problems that are not fully addressed by existing security technologies for distributed systems. For example, parallel computations that acquire multiple computational resources introduce the need to establish security relationships not simply between a client and a server, but among potentially hundreds of processes and associated protocols that implement this policy.

At the highest level, the security requirements of any system involve the unauthorized disclosure or modification of data and ensuring the continued operation of the system. Systems differ in the policy that determines when disclosure or modification is authorized, and they also differ in the kinds of attack to which they are subjected. Because grids typically span multiple organizations, and even different countries with different laws, security requirements may vary from one part of such system to another.

For most systems, however, including grids, security requirements encompass authentication and authorization.

We will discuss grid security requirements and provides an overview of some of the technologies that are available or under development to address these requirements. Latter, we will propose a dynamic and flexible security framework.

2. Security protocols

There are a vast number of security protocols to protect against various scenarios, including eavesdropping, man-in-the-middle attacks, and message alteration. All of them provide routines for encrypting/decrypting a message for online transmission and for authentication/authorization.

Because of the heterogeneous characteristic of grid computing these mechanisms have to be extremely flexible.

There are two important approaches in describing a security protocol:

- one based on public-key cryptography
- one based on shared-private key cryptography.

There two protocols that seems to be globally accepted: Kerberos and SSL.

The advantages of one of them are disadvantages for the other one. I will discuss these pros and cons in a further section. In a short comparison, the main difference between these two protocols is the manner in which is addressed the authentication problem: Kerberos use symmetric cryptography, SSL use asymmetric cryptography. The principal disadvantage of asymmetric cryptography is its performance Existing asymmetric cryptosystems(RSA,DSA) are significantly slower than their symmetric counterparts(RC4,DES). Thus, the both protocols use a symmetric algorithm for transmit effective data.

2.1 Kerberos

Private key encryption requires much less computing power than public key techniques and is usually faster, but it requires a shared secret that must be distributed in some other manner (out of band). For a small number of users, keys can be physically distributed using, for example, secure printing techniques. Often private keys are distributed using a public key technique.

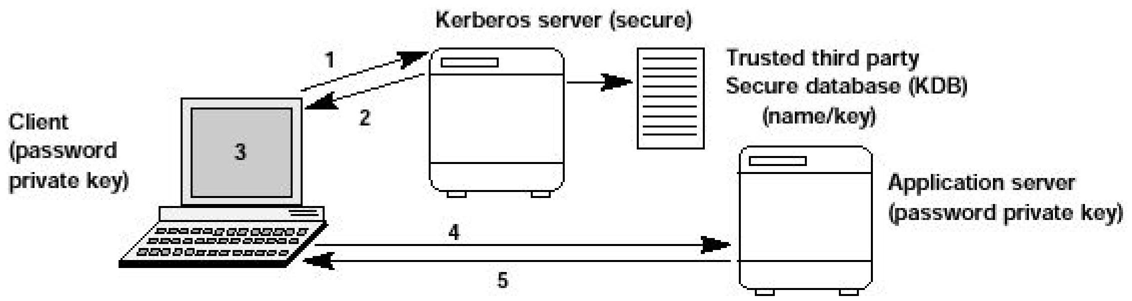
Kerberos is a private key encryption approach developed at the Massachusetts Institute of Technology in the mid 1980's as part of the Athena project. Its primary goal is to prevent plain text passwords from being sent across that network. Kerberos is a mechanism for authenticating workstation requests securely on an insecure network.

It is based upon a central (possibly replicated) security server that is trusted by everyone. Every user has his or her own private key that is shared only with the security server. This setup keeps you from having private keys for each user/server pair.

Kerberos is basically a trusted third party authentication service. This means that service providers can be assured that requests have come from where and who they appear to come from. The Kerberos scheme of things consists of *workstations*, *server providers* (servers) and *authentication and ticket granting server (AS,TGS that comprise the Key Distribution Server)* and a network. Effectively every request for a service by a workstation will include a user-id and password, this is transparent to the user and the information is not susceptible to network snooping.

The authentication messages that pass between the workstations and the authentication server are known as *tickets*, they are usually encrypted when they are known as *sealed tickets*. Of course such a scheme is only as good as the encryption mechanism and the security of the authentication server.

A basic Kerberos authentication session is as follow:



1. [Client]

-User enters password in response to login prompt.

-Constructs message composed of user name and the TGS server name

-Sends message to Kerberos Server (Authentication Service)

2. [KDC(Authentication Service)]

-Receives message

-Allocates arbitrary TGS session key

-Constructs ticket composed of:

{user-name, TGS-server-name, WS-net-address, TGS-session-key}

-Ticket is encrypted using TGS key which is only known to TGS. This is now the *sealed ticket*

-Constructs message composed of *{TGS-session-key, sealed ticket}*

-Encrypts message using user key which is normally the encrypted user password.

-Sends message to client.

3. [Client]

-Received encrypted message

-User enters password in response to prompt

-Message decrypted - workstation has now got TGS-session-key and sealed ticket

After this steps the client has a TGT (ticket-granting ticket). He will use this ticket in any further service access request. The ticket that can only be read by the TGS which will do so to validate requests for other services. Requests for services require that an authenticator be included with the request.

Next the client will make a securely request for a service.(SRV)

1' .[Client]
-Constructs an authenticator consisting of *{user-name, WS-net-address, time}*
-Encrypts authenticator with TGS-session-key yielding sealed authenticator
-Constructs a message consisting of:
{sealed-ticket, sealed-authenticator, SRV-name}
-Sends message to TGS

2' .[TGS]

-Received message
-Decrypts sealed ticket using TGS-key
-Decrypts sealed authenticator using TGS-session-key which was obtained from the decrypted sealed ticket;
-Validates user-name and WS-net-address by comparing values from authenticator and ticket. Also validates time preventing "play-back" attacks.
-Construct SRV-session-key
-Create SRV-ticket consisting of
{user-name, SRV-server-name, WS-net-address, SRV-session-key}
-Seals the SRV-ticket using the SRV server key which is known only to the TGS server and the SRV server.
-Constructs a message consisting of *{SRV-session-key, sealed SRV-ticket}*
-Encrypts the message using the TGS-session-key
-Sends the sealed message to client

3'. [Client]
-Received message
-Decrypts message using TGS-session key yielding SRV-session-key and the sealed SRV-ticket
-Constructs a SRV-authenticator consisting of:
{user-name, WS-net-address, time}
-Encrypts the SRV-authenticator using the SRV-session-key yielding a sealed SRV- authenticator.
-Constructs a message consisting of

{sealed SRV-ticket, sealed SRV-authenticator, SRV-server-name}

4. [Client]

-Sends the message, unencrypted to the SRV server

5. [Service SRV]

-Receives the message

-Decrypts the sealed SRV-ticket using the SRV-server-key known only to the SRV server and the
TGS server. From this it obtains the SRV-session-key.

-Decrypts the sealed SRV-authenticator. If the timestamp is recent, the server knows that the
essage was recently generated someone who knew the session key. Since the session key was issued (by TGS) only to the user named in the ticket, the client is authenticated

-If the client requires authentication from the server, the server extracts the timestamp, re-encrypts
it using the session key, and returns it to the client;

At this stage both the client and the SRV server are in possession of the SRV-session-key and
may use it to encrypt traffic between the two using a symmetric algorithm.

As a review, the following points should be noted:

- Kerberos depends on authenticators and tickets.
- tickets held on a workstation cannot be decrypted on the workstation.
- tickets are associated with session keys that are generated by a random mechanism when the ticket is generated.
- tickets are reusable, tickets usually have a lifetime of 8 hours.
- authenticators are not reusable and must be created afresh for each connection to a server.
- A server history of recent requests should result in the detection of duplicate requests due to stolen tickets.
- Tickets and authenticators contain network addresses so stolen tickets can't be used elsewhere without address spoofing.
- Clients may validate servers by requesting that the server return a message containing the client's time-stamp incremented by one. This message should be encrypted with the server-client session key. A fake server would not know the server decryption key so could not obtain the server-client session key.

2.2 Public key based protocols. SSL/TLS

Secure **S**ockets **L**ayer is a security protocol that has three basic properties:

1.The connection is *private*. Encryption is used after an initial handshake to define a secret key. Symmetric cryptography is used for data encryption (e.g., DES, RC4)

2.The peer's identity can be *authenticated* using asymmetric, or public key, cryptography (e.g.,RSA, DSS).

3.The connection is *reliable*. Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g., SHA, MD5, etc.) are used for MAC.

The SSL protocol runs at Network layer, above TCP/IP and below higher-level application protocols such as HTTP or IMAP.

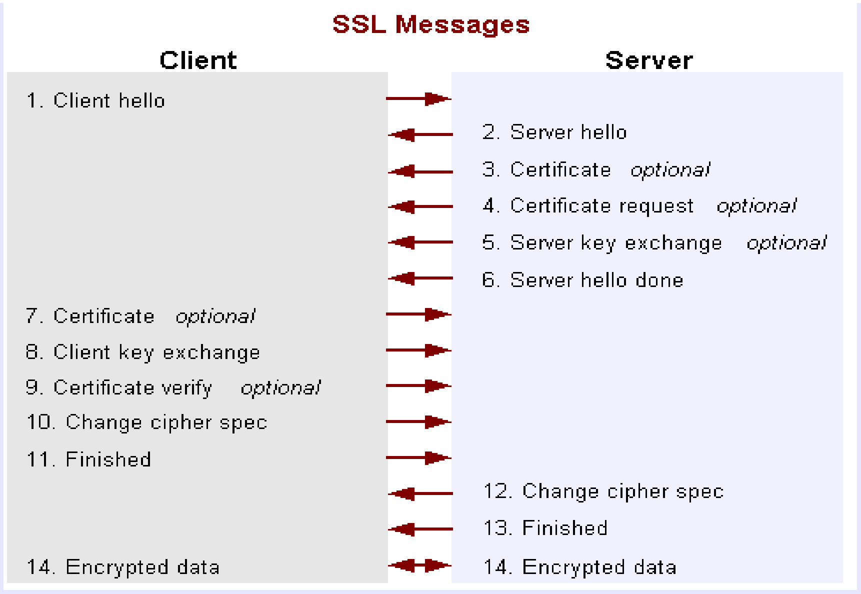
The protocol is composed of two layers: SSL Record Protocol and SSL Handshake Protocol.

At the lowest level, layered on top of some reliable transport protocol (e.g. TCP), is the SSL Record Protocol. The SSL Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the SSL Handshake Protocol, allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. One advantage of SSL is that it is application protocol independent. A higher level protocol can layer on top of the SSL Protocol transparently.

Basically SSL Protocol operates in two phases: in the first phase (handshake), using a public key technique, peers establish some common information, such encryption/integrity algorithms and a cryptographic secret key. In the second phase, peer communicate securely using a symmetric algorithm and reliable using a secured hashing algorithm (that produce a Message Authentication Code (MAC)).

The figure below shows the sequence of messages that are exchanged in the SSL handshake. (also knows as SAP: SSL Authentication Protocol). Messages that are only sent in certain situations are noted as optional:

The SSL messages are sent in the following order:



1.*Client hello* - The client sends the server information including the highest version of SSL it supports and a list of the cipher suites it supports. (TLS 1.0 is indicated as SSL 3.1.) The cipher suite information includes cryptographic algorithms and key sizes.

2.Server hello - The server chooses the highest version of SSL and the best cipher suite that both the client and server support and sends this information to the client.

3.*Certificate* - The server sends the client a certificate or a certificate chain. A certificate chain typically begins with the server's public key certificate and ends with the certificate authority's root certificate. This message is optional, but is used whenever server authentication is required.

4.*Certificate request* - If the server needs to authenticate the client, it sends the client a certificate request. In Internet applications, this message is rarely sent.

5.Server key exchange - The server sends the client a server key exchange message when the public key information sent in 3) above is not sufficient for key exchange.

6.Server hello done - The server tells the client that it is finished with its initial negotiation messages.

7.Certificate - If the server requests a certificate from the client in Message 4, the client sends its certificate chain, just as the server did in Message 3.

8.*Client key exchange* - The client generates information used to create a key to use for symmetric encryption. For RSA, the client then encrypts this key information with the server's public key and sends it to the server.

9.Certificate verify - Its purpose is to allow the server to complete the process of authenticating the client. When this message is used, the client sends information that it digitally signs using a cryptographic hash function. When the server decrypts this information with the client's public key, the server is able to authenticate the client.

10.*Change cipher spec* - The client sends a message telling the server to change to encrypted mode.

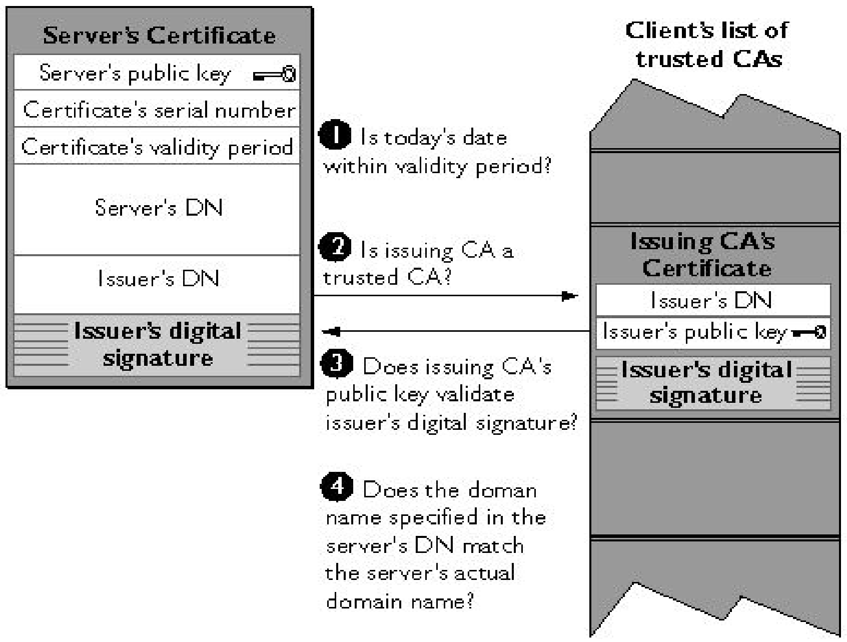
11.Finished - The client tells the server that it is ready for secure data communication to begin.

12.*Change cipher spec* - The server sends a message telling the client to change to encrypted mode.

13.*Finished* - The server tells the client that it is ready for secure data communication to begin. This is the end of the SSL handshake.

14. Encrypted data - The client and the server communicate using the symmetric encryption algorithm and the cryptographic hash function negotiated in Messages 1 and 2, and using the secret key that the client sent to the server in Message 8.

SSL Handshake protocol permits peers to authenticate each other using a public key technique based on X509 certificates. The following pictures explain in detail this


Server's Certificate
Server's public key
Certificate's serial number
Certificate's validity period
Server's DN
Issuer's DN
Issuer's digital signature
Client's list of trusted CAs
1 Is today's date within validity period?
2 Is issuing CA a trusted CA?
3 Does issuing CA's public key validate issuer's digital signature?
4 Does the doman name specified in the server's DN match the server's actual domain name?
Issuing CA's Certificate
Issuer's DN
Issuer's public key
Issuer's digital signature

An SSL-enabled client goes through these steps to authenticate a server's identity:

process:

Authenticate the server to the client:

An SSL-enabled client goes through these steps to authenticate a server's identity: (step 4 is optional and is not included in standard protocol but protect the peers against the man-in-the-middle attack)

1. Is today's date within the validity period? The client checks the server certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to Step 2.

2. Is the issuing CA a trusted CA? Each SSL-enabled client maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 2. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not on the list, the server will not be authenticated unless the client can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details).

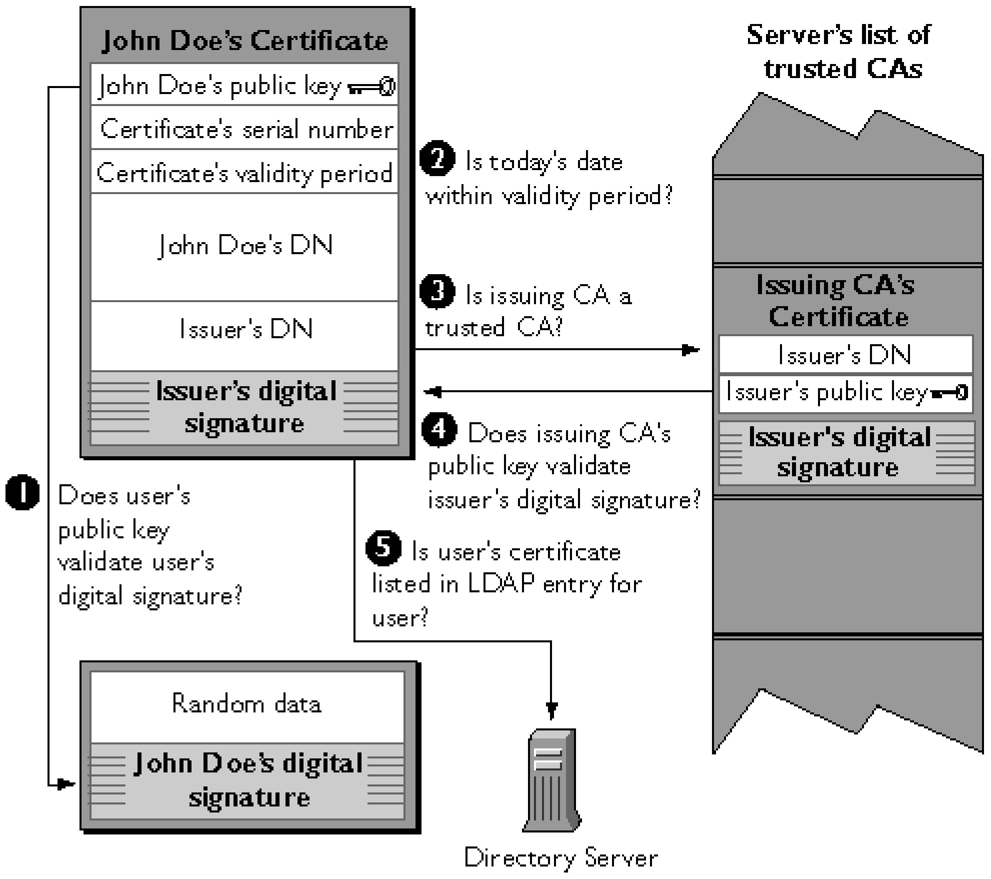
3.Does the issuing CA's public key validate the issuer's digital signature? The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.

4. Does the domain name in the server's certificate match the domain name of the server itself? This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as a Man-in-the-Middle Attack. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to Step 5.

5. The server is authenticated. The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user will be warned of the problem and informed that an encrypted and authenticated connection cannot be established.

Authenticate the client to the server:

An SSL-enabled server goes through these steps to authenticate a user's identity:


John Doe's Certificate
John Doe's public key
Certificate's serial number
Certificate's validity period
John Doe's DN
Issuer's DN
Issuer's digital signature
Server's list of trusted CAs
Issuing CA's Certificate
Issuer's DN
Issuer's public key
Issuer's digital signature
1 Does user's public key validate user's digital signature?
2 Is today's date within validity period?
3 Is issuing CA a trusted CA?
4 Does issuing CA's public key validate issuer's digital signature?
5 Is user's certificate listed in LDAP entry for user?
Random data
John Doe's digital signature
Directory Server

1. Does the user's public key validate the user's digital signature? The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed.

At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.

2. Is today's date within the validity period? The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes on to Step 3.

3. Is the issuing CA a trusted CA? Each SSL-enabled server maintains a list of trusted CA certificates, represented by the shaded area on the right side of Figure 3. This list determines which certificates the server will accept. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is

yes, and the server goes on to Step 4. If the issuing CA is not on the list, the client will not be authenticated unless the server can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details). Administrators can control which certificates are trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. Does the issuing CA's public key validate the issuer's digital signature? The server uses the public key from the CA's certificate (which it found in its list of trusted CAs in Step 3) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated.

5. Is the user's certificate listed in the LDAP entry for the user? This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The server can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step will then refuse to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake.

2.3 Disadvantages/advantages

1) Secret-key security architectures use symmetric encryption and decryption techniques. These techniques rely on one secret key that is known to all authorized communication partners. The required distribution of secret keys over potentially insecure networks before a secure communication can be established is the major issue with secret-key cryptography. The relative simple and computationally inexpensive encryption and decryption algorithms define its advantages.

Kerberos used alone or under the distributed computing environment authenticates users through a secure transaction with a centrally maintained key server. Kerberos achieves interorganizational, or cross-realm, authentication by designating trustworthy key servers in other organizations. Kerberos meets many of the basic requirements for virtual organization authentication, but it presents two problems.

1. Using Kerberos for intersite authentication also means using it for intrasite

authentication, which is often not feasible because of equipment and staffing costs. Sites must negotiate many cross-realm authentication agreements

2. It requires site-based trust relationships, rather than user-based trust relationships. The fact that a user already has trust relationships with multiple sites (i.e. the user can login to each Kerberos realm) is not sufficient under Kerberos to allow that user to use resources at multiple sites as part of a single, secure, distributed operation. For this to work, the Kerberos security administrators of those two realms must set up inter-realm trust agreements. History has proven that this not feasible in practice except in the case of tightly controlled Grids such as in the military and other classified networks.

2). **SSL(TLS)** has two major advantages over Kerberos: (1) It doesn't require an accessible trusted third party; (2) it can be used to establish a secure connection even when one end of the connection doesn't have a "secret" ("key" or "password"). These two advantages make it ideal for secured communication and for applications where there is a large user base which is not known in advance.

3). Both SSL and Kerberos require a permanent online server (CA revocation list server respectively KDC server). So this may be considered an disadvantage for both protocols since these servers must present a high level of availability, security since breaking them means compromise the entire security.

The advantages of secret-key and public-key cryptography can be combined in a hybrid architecture: public-key methods are used for connection set up and authentication in order to solve the key distribution problem. Once a secure channel has been established, symmetric session keys are generated, exchanged and subsequently used in faster secret-key encryption of the exchanged messages.

2.4 Kerberos-SSL accommodation

2.4.1. PKINIT

PKINIT is an extension of the Kerberos protocol (RFC1510), which allow users with public key certificates to use them in initial authentication. The basic mechanism is as follows: The user sends a request to the KDC as before, except that if that user is to use public key cryptography in the initial authentication step, his certificate and a signature accompany the initial request in the preauthentication fields. Upon receipt of this request, the KDC verifies the certificate and issues a ticket granting ticket (TGT) as before, except that the TGT is now encrypted utilizing either a Diffie-Hellman derived key or the user's public key. This message is authenticated utilizing the public key signature of the KDC.

2.4.2. Online CA (K5cert).

This service creates users certificates on-the-fly. This service has its own signing key and corresponding public key certificate and acts as a CA. A new certificate is created for each client s request for credential. Lifetime of such a certificate is usually short (12 hours). Before accessing the Repository, the client generates a new key pair, and then a certificate request using the new public key. The identity of the Repository is used as the certificate issuer, while the subject name is derived from the client's ID that is used for authentication to the Repository. This request is sent to the Repository who then signs it with the Repository's signing key. The newly created certificate is sent back to client. The K5cert implementation uses Kerberos as an authentication mechanism.

3. Security Requirements in Grid

In traditional systems, the focus of security mechanisms has been to protect the system from its users and, in turn, to protect data maintained by the system on behalf of one user from compromise by another. While such protection remains important for grid applications, grids introduce the extra requirements of protecting applications and user data from the systems on which parts of a computation will execute. Further, because running code may originate from many points on a network, there is greater potential for running malicious code, requiring stringer methods to verify the origin and authenticity of the code.

In any particular grid, there is a multitude of hardware/OS configurations, potentially owned by different corporations, probably in multiple countries(with different laws), all providing resources to multiple communities of users, all of whom probably have different security policies. Thus, security requirements may vary from one part of such system to another.

Virtual Organizations (VOs) are created when computational and intellectual resources from separate physical organizations are shared to solve problems that require the combined efforts and resources of the VO members (e.g., collaborative problem solving). The resources shared in a VO include data files, computers, software, and specialized resources such as scientific instruments. In a such virtual organization there are two kind of actors, users and resource providers. To make possible interaction between them a standard security mechanism must exist.

In any system where security must be assured, the following requirements must be meet:

Authentication - mechanism for establishing the identity of a user or resource;

Authentication is the process of establishing the identity of a participant to an operation or request. A principal is an entity whose identity is verified through authentication and on whose authority the operation is performed or authorized. The principal may be the user logged into a remote system and on whose behalf the application client is running, it may be local user logged into a server, or it may be the server its self.

In traditional systems, the requirement for authentication is focused on the client, since the goal of security in such systems is to protect the system (servers) from the users. In grid systems, mutual authentication of the server is just important, to ensure that resources and data provided by the server are not really provided by an attacker. User and server authentication provides assurance that the principal or more precisely a process possessing some objects or secret held or known by the principal, is an active participant in a protocol exchanged at the authentication is performed.

Data origin authentication provides assurance that a particular message, data item, or executable object originated with a particular principal, and makes it possible to determinate the origin of an incoming program. This information can be used to determinate whether a program was modified or was sent by an attacker to compromise the resources to which the program has access. By itself, however, data origin authentication does not ensure that the data was recently sent by the principal, only that it was generated by the principal at some point in the past. In some cases, an application or process may assume the identity of a different for the purpose of performing particular operation. Authority to act as this other principal is granted through a process called delegation identity.

Authorization - mechanism for determining whether an operation is consistent with the defined sharing relationships.

Authentication is useful primarily to enable authorization. Authorization is the process through it is determined a particular operation is allowed. In traditional systems, authorization is usually based on the authenticated identity the request and on information local to the server. This local information identifies the individuals authorized to perform an operation, and it often takes the form of an access control list associated with a file, directory, or service. Authorization mechanisms are required within grid systems to determine whether access to a resource is allowed. Such resource access may involve accessing a file on a data repository, reserving network bandwidth by using a system like RSVP, or running a task on a particular processing node. In some cases, the ability to run a task on a processing node may be identity of the user asking to run the task, but on the identity of the task or application to be run. When the code for an application be determined from the name of the application; but if the user provides the code to be run, the application itself must be authenticated by using data origin authentication - usually by verifying a digitally signed checksum of the executable. To identify the particular programs, access control list might contain the names or checksum of authorized programs, together with the names of principals authorized to invoke the program.Many applications can benefit from an authorization mechanism that supports delegation of authority. Delegation of authority is a means by which a user process authorized to perform an operation can grant that authority to perform the operation to another process. This is a more restricted from of delegation that delegation of identity (discussed earlier). Delegation of authority is important task that will run remotely on the grid but that must make calls themselves to read or write data stored across the network. For example, in implementing distributed authorization, a resource manager might allocate a node to a job and might delegate to the job's initiator the authority use that node.

Assurance

While authorization mechanisms allow the provider of a service to decide whether to perform an operation on behalf of the requester of the service, assurance mechanisms allow the requester of a service to decide whether candidate system or service provider meets the requester's requirements for security, trustworthiness, reliability, or other characteristics.

Assurance is a form of authorization used for validating the authority of the service provider. When applied to computer systems, this authorization of the system for use in particular application is sometimes called accreditation. In a computational grid , assurance credentials may be checked when selecting nodes for a computation, to ensure that they meet the performance, reliability, and security requirements of the application and that the computing service is run by an organization that is trusted to handle the data used by task what will run on the selected nodes. When applied to programs, a resource manager might verify assurance credentials attached to a program before it is run.

Accounting

Accounting provides the means to track, limit, or charge for the consumption of resources in a system. It is critical for providing a fair allocation of the available resources to users that need them. Accounting will be critical for deployment of grid applications, supporting payment or barter for the use of computing resources and providing incentives to the owners of computing resources to make idle capacity available to others. Additionally, when the aggregate computing requirements of grid applications exceed available resources, the accounting system will provide a tool that is useful in deciding which processes to run. Any grid accounting mechanism must be distributed so that quotas can be applied to any node in the grid, making the allocation of resources more flexible than it would be of quotas were maintained separately on each node. Further, because computational grids will cross organizational boundaries, the accounting servers should be distributed and scalable across administrative domains. This feature will allow an organization to administer quotas for its users, independently from the quotas granted and maintained by other organizations. To prevent the compartmentalization of the computing resources in these domains, a settlement and clearing process should be provided between accounting servers in different domains.

Audit

An audit function records operations that have been performed by a system, associating each action with the principal on whose behalf the operation was performed. An audit is useful for figuring out what wrong if something breaks or for tracking breaches of security in the system. An instruction detection system will look at events generated by the audit in order to find patterns of operation that fit the profile of a system instruction or that do not fit the profiles of legitimate users. If such patterns are detected, an alert is generated. In traditional systems, the audit function is local to each server. To detect network attacks, the audit function should be distributed or audit records transmitted to a central location for each organization or administrative unit, where a higher-level view of the system can be constructed. In the ideal case, summary information(and in certain cases details) might be shared across administrative boundaries. Because code can be loaded onto the system nodes from many sources, and because grid computations can take place across multiple nodes, an audit mechanism for a grid must itself be distributed. Consider the case of a denial-of-service attack on the grid. To be effective, the attack would have to be mounted across many nodes. A distributed audit function could aid in identifying such an attack.

Access policy

Resource providers and users must define clearly and carefully what is shared, who is allowed to share, and the conditions under which sharing occurs;
Access control mechanisms have to be:

* scalable and reliable - Ability to manage increase in users and resources as collaborations between other organizations increase "·(e.g. avoid single point of failures, increase availability though redundancy without compromising security)

* manageable and maintainable - adding, removing and modifying user privilege need to be kept easy and intuitive

* preferably under the control of the resource providers - Organizations prefer to have control over who have access to their data.

Secure communication (integrity, privacy) – data exchanged must flow in a secure manner between endpoints.

The most direct application of cryptography to security is to protect the confidentiality of data accessed though computer networks. When the sender and receiver share an encryption key known only to them, data can be encrypted before transformation and decrypted after transmission, protected the data from disclosure to eavesdroppers. Encryption is the only suitable means to provide confidentiality and integrity of data as it is transmitted across an open computer network such as that which connects the nodes of any largescale, administratively decentralized computational grid.Besides protecting the confidentiality of data, encryption also protect data integrity. Because knowledge of the encryption key is required to produce ciphertext that will yield a predictable value when decrypted, modification of the data by someone who doesn't know the key can be detected by attaching a checksum to the data before encryption and requiring that the receiver verify the checksum after decryption. Alternatively, a message digest function can be calculated over data sent unencrypted, but the digest itself is encrypted and attached to the message to provide a digital signature. A message digest function is a one-way function used as a checksum: given a message and a digest, it is computationally infeasible to find a different message that shares the same digest.

Certification

Encryption provides the base technology for confidentiality and integrity of data communications, and authentication methods for distributed systems allow the user to prove possession of an encryption key known only by the user, but it is the certification mechanism that provides the binding between a particular encryption key and the authentication identity. A certification authority (CA) is the third party that certifies this binding, issuing a certificate signed by the CA that attests to the validity of the binding.

A certificate is a data object that specifies a distinguished name of a principal an, for certificates based on public key that was issued to or selected by the principal and that is the inverse of the private key known only by the principal. The certificate may contain additional attributes of the principal; depending on the kind of certificate, these might include authorizations, group membership, email addresses, or alternate names. Once constructed, the certificate data is signed by the CA, ensuring its authenticity. X.509 [509] is the most widely used certificate format; X.509 certificates are used by most web browsers, commercial secure email products, and public-key-based electronic payment systems

To validate the binding of the key in the certificate to a distinguished name and to other attributes, the verifier must validate the CA's signature.

This validation requires knowledge of the CA's public key. The key may be known a priori by the verifier, or it may be obtained from the CA's certificate, which was its self issued by a higher-level CA. Thus, certification is usually hierarchical, with CAs authorized to issue certifications only for distinguished names delegated by the higher-level CA.

Clients are configured with certain well-known public key that are in many cases obtained when software is installed. These keys are used to validate the certificates lower-level Cas, whose keys are then used to validate the certificates for end users and other Cas. Because certificates are used for many purposes, and because of the lack of a single universally trusted certification authority, the certification hierarchies used in practice have multiple roots, and applications and servers are configures with the public keys of those root nodes whose certifications are trusted.

Global identities in grids

Since the local security mechanism (that perform authentication and authorization) vary from site to site in a grid environment a solution to implement a global infrastructure is to define a common way of expressing global identities (users or resources) used in security protocols. Hence, it is imperative to employ a standard (such as X.509v3). These global identities are used for inter-domain communication. Access to local resources will typically be determined by a local security policy that is enforced by a local security mechanism. It is impractical to modify every local resource to accommodate interdomain access, so, there must exist a mapping method between global to local subjects(operating systems accounts, Kerberos credentials, etc).

As a conclusion Grid environments demand sophisticated security features to gain the trust of both, the users who want to run their programs protected from unauthorized interference and the resource owners, who demand authoritative autonomy over their machines. In many systems the following features must exist:

-Single sign-on: a user should only need to authenticate to the system once per session (e.g. once per work day) ·

-Protection of credentials: user credentials should not be exchanged over insecure networks

-Support for secure group communication and management

-Ability to adapt to legal export regulations, e.g. through the use of unregulated encryption methods

-Support for multiple implementations on different architectures

-Support for restricted delegation of rights to other entities/processes

-Role support (an entities rights need to adapt to his current role / group membership)

4. Existing Grid Security Solutions

4.1 Systems that provides security for distributed applications at nework layer:

IPSec, Ipv6 and VPNs

Many of the attacks on the security of distributed systems rely on the ability of an attacker to monitor and modify packets on the network. The IPSec suite of protocols developed by Internet Engineering Task Force (IETF) and the security services that are present in IP version 6 provide for confidentiality and integrity protection of data at the network layer when sent between end systems. When communication is first established between a pair of Internet hosts, a key distribution is initiated of exchange a conventional encryption key.

That key is used to provide confidentiality and integrity of the packets subsequently exchanged between the two systems. The key distribution function may be based on public-key cryptography, it may be based on other key distribution mechanisms like Kerberos, or it may use keys that were distributed in advance between the communicating systems. In contact to the other examples of authentication and key distribution, these keys are associated with the communicating hosts rather than with applications or end users.

IPSec, Ipv6, and propriety technologies available from some vendors allow the creation of virtual private network (VPNs), networks implemented by using the shared physical infrastructure of the Internet but with communication permitted only between participating nodes in the private network and where communication is protected from disclosure to and modification by nodes that are not participant. These systems provide some improvement in security for distributed applications and will often be the appropriate technologies to use when it is impractical to integrate security at the application layer (which might be difficult without the source code for the distributed application).

However, because these systems operate at the network layer, they cannot provide for authentication of the end user, and they do not have knowledge of the application-level objects that are to be protected. Hence, they have limited ability to support security polices that distinguish users application objects.

Firewalls

Firewalls provide barrier at the boundary to an organization's network through which only specifically authorized communication may proceed. In general, firewalls fill an important need in an organization's security policy because of they have been properly configured and if all paths into the network are protected by a firewall, they prevent many kinds of attack on hosts within the organization's network. Firewall are less useful as a means to protect grid applications because the communication pattern for legitimate application running on a computational grid will, by their very nature, require communication through the firewall, making it difficult for the firewall to distinguish legitimate communication from security violations. By integrating IPSec and VPN technologies at network boundaries, firewalls can play a role in constructing a computational grid across a set of cooperating organizations. In such a system, communication on the internal network of the cooperating organizations could remain unprotected. A firewall at the boundary between each unprotected network and the rest of the Internet would encrypt message leaving the local network and decrypt message entering the local network. Communication between nodes in this private grid shared by cooperating organizations would then be protected when sent over the Internet, but would remain in the clear for communication within the local network, hence removing the need for each internal host to maintain its own set security parameters.

Integration with communication layers

For the services described so far to have an effect on the security of a computational grid, the protocol already developed and those under development must be integrated with the communications and resource management mechanisms used by the grid. In general, integration is one of the most difficult aspects of deployment security services today. Security services can be integrated with protocol at several layers. Effects are under way in the IETF to add security services at the IP layer [33]. With these extensions, computer systems will be able to authenticate to one another, and communication between the systems can be encrypted. Integrating security services at this layer does not provide authentication of the individual users of the system to the remote service providers and thus does not, by itself, meet the requirements for authentication (in support of access control) by many applications. It does, however, improve the confidentiality and integrity of communications by applications running on those , including applications that have not been modified to use application-level security services.

Integration of security services can also occur at the application layer, and change at the application layer are necessary for services where the operations allowed depend on the identity of the user. Integrating security at this layer can be cumbersome, requiring changes to the application protocol for each application. The Common Authentication Technology Working Group of the IEFT has developed the Generic

Security Services Application Programming Interface (GSS-API) to facilitate the integration of security services at the application layer.

When using the GSS-API, application make calls to authentication, confidentiality, and integrity in a manner that is independent of the underlying security services.

Integration of security services is easier for applications that run on top of RPC and similar transport mechanisms. When running on top of such transport protocol, user authentication, confidentiality, and integrity can be provided at the transport layer. Through the application must still be modified to ask the right questions and to use the answers as basis for authorization, such changes to the application are less intrusive than changes to the application protocol itself. Security services have been integrated at the RPC layer for the Open Software Foundation's DCE RPC [425], and Sun's ONC RPC.

The transport layer is likely to be the correct place to integrate security services for a computational grid. Security services can be integrated with the communications layer used for communication between cooperating tasks, providing the appropriate level of communications security (confidentiality and integrity protection) for the application's needs. The level of protection provided at this layer may be adjusted as appropriate also to take into account knowledge about the lower-level communication-medium. For example, when two tasks are communicating across a bus on a tightly coupled multiprocessor, where it is known that no untrusted jobs have access to the bus, encryption might be bypassed, improving the performance of the communication primitives.

4.2. Systems that provides security for distributed applications at application layer.

4.2.1 Grid Security Infrastructure (GSI)

The Grid Security Infrastructure (GSI) is a public key system for mutual authentication and authorization of grid users, processes and resources. It is an important component of the Globus Toolkit but can also be used independently. The GSI relies on standardized X.509v3 certificates for authentication and access control lists (ACLs) for authorization. An access request is authorized if a resource has an entry in its grid map file (its ACL) for the (authenticated) global identity presented with the request. The global user-id is then mapped to a corresponding local user-id and finer grain authorization is left to the local resource operating system mechanisms. This allows the GSI to be layered securely on top of existing systems and to provide uniform credentials and certification infrastructure without undermining local security mechanisms.

Single sign-on functionality is achieved through the use of proxy certificates (PC). A proxy certificate is derived from a user certificate (end entity certificate) or another PC. A PC is used to delegate the user's authority to a process for a very limited time period (usually a work day). The proxy process can then authenticate on the user s behalf to remote resources and processes. This concept is comparable to session keys in secret key systems, the proxy certifies the temporary binding of a session key pair to the user s identity. Access to the confidential user credentials is only required at initial log-on when a proxy is created. Through the use of proxy certifications, the principle long lived user credentials do not have to be used directly when authenticating to a remote resource and can thus be more easily protected against misuse. The credentials of a proxy certificate do not require the same level of protection as they are only valid for a short amount of time (the session).

The GSI accesses security functions through the Generic Security Services API. This makes the GSI independent of specific encryption algorithms and enables it to easily adapt to varying legal regulations (e.g. export of encryption technology). The GSSAPI is available for different platforms which provides for portability. The current implementation of the GSI uses the SSL/TLS handshake protocol (SAP) for authentication of communicating parties. The SSL/TLS protocol also provides for message protection.

The GSI released with the Globus Toolkit 3 (GSI3) is based on and largely compatible with the earlier GSI implementations but has been improved and augmented with new features aimed at securing a services based architecture. The new features support Web Services Security specifications, such as SOAP [BOX00] with XMLSignature and XML-Encryption for authentication and message protection.

Context establishment is based on the proposed WS-SecureConversation protocol. Furthermore, a new format for proxy certificates has been implemented. The new format supports the use of policy statements in proxy certificates to restrict the set of delegated rights.

Limiting the set of delegated rights improves security in the case of a stolen proxy certificate and private key and enables the more selective use of rights. The same mechanisms are leveraged to enable client-side authorization.

A service using GSI3 can hold a proxy certificate (delegated from the certificate of the host where the service is running) which specifies the service owner, the local account under which the service executes and local resource policy that applies to the service in the policy restrictions. This enables clients to verify on which hosts their services execute and under which local user identity.

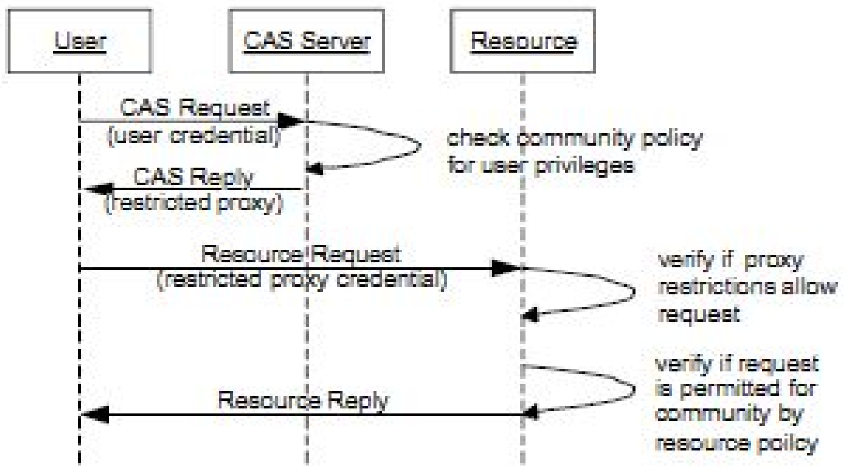
4.2.2 Globus Community Authorization Services (CAS)

The CAS model is a very recent approach at managing the assignment of access privileges to users from several communities. It is based on the GSI and uses restricted proxy certificates to delegate rights. CAS allows a resource provider to assign coarse-grained access rights for its resources to a whole community (e.g. a virtual organization corresponding to applicable acceptable usage policies (AUPs). A community uses CAS servers as a centralized trusted party within the community to specify fine-grained access policies for each community member. Administrative requests provide for maintenance of the community policy database. The CAS authorization mechanism supplies policy information from this database in the form of signed credentials to the user which in turn groups them with his request to a resource. Authorization mechanisms at the resources need to enforce the stated policies. Scalability is achieved by using existing access-control mechanisms (e.g. user logins) only to determine community membership (e.g. through group accounts). This will enable large scale virtual organizations with many collaborators and resources as the number of accounts that a resource administrator needs to manage is equivalent to the number of communities and not the number of community members. For fine-grained authorization decisions the Generic Authorization and Access Control API (GAA-API) is employed. CAS enabled applications have to use the security context of this API; The general scheme of authentication and authorization in CAS is presented in figure 4.2.2.
Resource owners grant coarse-grained privileges for their resources communities ;
CAS servers maintain fine-grained access control information and grant restricted proxy credentials to community members ;

Users make requests to the resources using the CAS proxy credentials; Resources grant access based on the resource's own access policy and the community policy stated in the proxy credentials.

An user's effective rights are the intersection of the coarse-grained rights granted by resource owner to the community and the more fine-grained rights granted by the community to the community member (the user).

Figure 4.2.2


User
CAS Server
Resource
CAS Request (user credential)
check community policy for user privileges
CAS Reply (restricted proxy)
Resource Request (restricted proxy credential)
verify if proxy restrictions allow request
verify if request is permitted for community by resource policy
Resource Reply

4.2.3 Permis

PERMIS is a privilege management infrastructure that uses a role based access control (RBAC) scheme that allows privileges to be associated with roles rather than with specific entities. Entities can hold several roles. X.509 attribute certificates securely bind and communicate role membership among the components of the infrastructure.

A central publicly accessible LDAP sever is used for hosting Attribute Certificates. Organization's Privilege Allocator create Authorization Certificates for users and stored in publicly accessible LDAP Directories. Also Authorization policy description are created and stored in publicly accessible LDAP directories. While querying a resource user presents its certificate . The Resource's Access Decision Framework retrieves the user's Attribute certificate and the policy definition from the LDAP server and enforces the privileges.

ACs are issued to users, and hold their privileges (either directly or indirectly via an attribute/role). Attributes comprise a type and value. PERMIS uses DER-encoded attribute certificates in X.509 standard format. A user is identified by his globally unique X.500/LDAP distinguished name, and a user has to be authenticated against that name. Attributes can form arbitrarily complex role hierarchies. The certificates are stored in LDAP repositories, using standard LDAP/X.500 schema. The base code can be extended to support other repositories with LDAP-based naming conventions.

4.2.4 EU Data Grid VOMS

VOMS classifies authorization information into two categories :

a) General information regarding the relationship between the user and the Virtual Organization;

b) Information regarding what the user is allowed to do at the Resource Provider Relationship between VO and user is specified as group and role by VOMS server (coarse grained) Information regarding what the user is allowed to access is maintained by the Resource provider. (fine grain)

The Virtual Organization Membership Service (VOMS) constitutes a system conceptually similar to CAS. It also has a community centric attribute server that issues subject attributes to members of the community. In VOMS however, the subjects authenticate with their own credentials (in contrast to a limited group credential in CAS) and subject attributes allow for the use of community privileges.

4.2.5 Akenti

Akenti is an access control system for widely distributed resources. Akenti uses public-key certificates for identification, authentication and authorization. Three types of certificates are used:

(1) identity certificates
(2) use-condition certificates
(3) attribute certificates.

For the identity certificates the X.509 standard certificate in ASN.1 format is employed, the other certificates are stored in an ASCII format proprietary to the Akenti system. Authorization of a request to a resource is handled by the Akenti policy engine, which compares attributes (e.g. group membership) of the authenticated user to use-conditions specified by the resource owners (called stakeholders).

Attribute certificates convey attributes to user identities and use-condition certificates specify the conditions that must be satisfied before a request is granted. Repositories are used to store certificates, these repositories can be implemented as LDAP, HTTP, or SQL servers as well as plain file systems. The resources use an authority configuration file to learn what repositories need to be queried, which certificate authorities they trust, and who is a stakeholder (i.e. who can issue a use-condition for the specific resource). Use-condition repositories are distributed to avoid centralized access policy information and distribute load but are typically kept close to the resources to reduce communication overhead.

5. MonaLISA

The MonALISA (Monitoring Agents in A Large Integrated Services Architecture) system provides a distributed service architecture which is used to collect and process monitoring information. While its initial target field of application is networks and Grid systems supporting data processing and analysis for global high energy and nuclear physics collaborations, MonALISA is broadly applicable to many fields of “data intensive” science, and to the monitoring and management of major research and education networks. |

5.1 Architecture

The MonaLISA (Monitoring Agents in A Large Integrated Services Architecture) system provides a distributed service for monitoring, control and global optimization of complex systems. MonALISA is based on a scalable Dynamic Distributed Services Architecture (DDSA) implemented using Java / JINI and Web Services technologies. The scalability of the system derives from the use of a multi-threaded execution engine to host a variety of loosely-coupled self-describing dynamic services or agents, and the ability of each service to register itself and then to be discovered and used by other services, or clients that require such information.

A service in the DDSA framework is a component that interacts autonomously with other services either through dynamic proxies or via agents that use self-describing protocols. By using dedicated lookup services, a distributed services registry, and the discovery and notification mechanisms, the services are able to access each other seamlessly. The use of dynamic remote event subscription allows a service to register an interest in a selected set of event types, even in the absence of a notification provider at registration time. The lookup discovery service will then automatically notify all the subscribed services, when a new service, or a new service attribute, becomes available.

The code mobility paradigm (mobile agents or dynamic proxies) used in the DDSA extends the approaches of remote procedure call and client- server. Both the code and the appropriate parameters are downloaded dynamically into the system. Several advantages of this paradigm are: optimized asynchronous communication and disconnected operation, remote interaction and adaptability, dynamic parallel execution and autonomous mobility. The combination of the service architecture and code mobility makes it possible to build an extensible hierarchy of services that is capable of managing very large systems.

The Monitoring Service

An essential part of managing a global system, like the Grids, is a monitoring system that is able to monitor and track the many site facilities, networks, and the many task in progress, in real time. The monitoring information gathered also is essential for developing the required higher level services, and components of the Grid system that provide decision support, and eventually some degree of automated decisions, to help maintain and optimize workflow through the Grid. MonALISA is an ensemble of autonomous multi-threaded, agent-based subsystems which are registered as dynamic services and are able to collaborate and cooperate in performing a wide range of monitoring tasks in large scale distributed applications, and to be discovered and used by other services or clients that require such information. MonALISA is designed to easily integrate existing monitoring tools and procedures and to provide this information in a dynamic, self describing way to any other services or clients. MonALISA services are organized in groups and this attribute is used for registration and discovery.

5.2 Security Requirements in MonaLISA

Monitoring modules in a MonaLISA service may collect sensitive data from the regional center it runs on. In this case, farm's administrators must have the possibility to define access policies in order to control the access. The confidentiality of data flow between services and clients must be assured too.

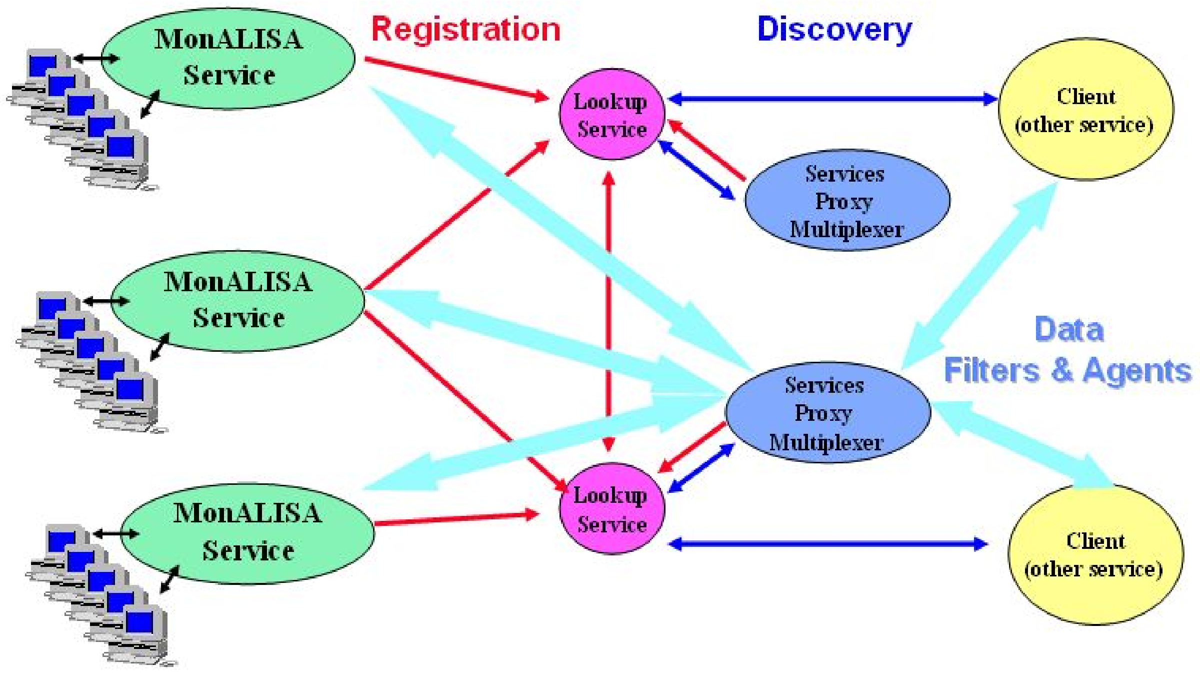
The security infrastructure proposed will rely on standardized X.509v3 certificates for authentication (since this format is common in actual grid systems), on access control lists (ACLs) for authorization and also on Secure Sockets Layer (SSL) communication protocol.

Also, the security infrastructure must support different types of authorization (e.g push model, push model, end entity certificate, GSI proxy certificates). Access control lists have to be manageable and maintainable (adding, removing and modifying user privilege need to be kept easy and intuitive).

5.3. Security Infrastructure

In the actual architecture clients interact with MonaLISA services via a the proxy service. Figure 5.4.1 describe clients-services communication.

Figure 5.4.1



The architecture provides a proxy service which is used by clients to connect to different services. The proxy service is also a JINI service. In our service design we use the mutual discovery between services and proxies to detect when a certain service runs run behind a firewall or NAT.

In this case the service initiates a connection to all the available proxies for a community and registers itself with the LUSs. Any client can interact now with such services via the proxy services. At the same time the proxy service does an "intelligent" multiplexing of subscribed data for multiple clients. We run multiple proxy services for redundancy and also for a dynamic load balancing of clients.

Since clients does not directly connect to MonaLISA services, a proper place for authorization decisions and authorization enforcement is the proxy itself. In order to achieve this, the MonaLISA service must delegates the authorization actions to proxy service. But, first of all, the proxy service must be trusted to do this on behalf of MonALISA service.

This can be assured by establishing a trust relationship with the proxy it connects to, using a public-key cryptography mechanism with X509 certificates. So, when MonaLISA service connects to a proxy, it must authenticate the proxy (verify that it received a trusted identity certificate).

Any further data request from a client is managed by the proxy service. It authorize this request based on authorization policy received from farm and the credentials presented by the client. Schematically this mechanism can be presented as fallows:

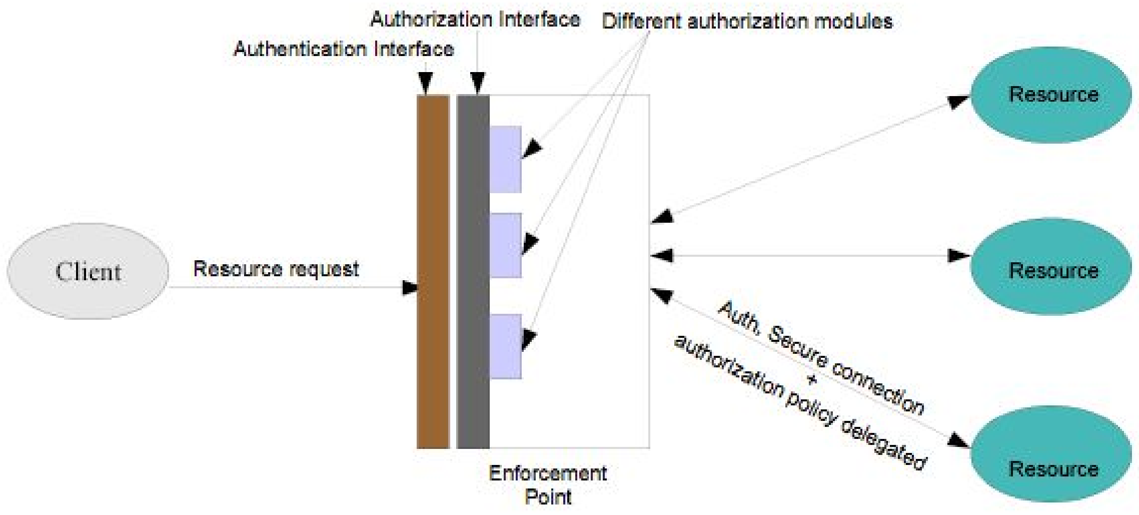


Fig. 5.4.2
Client, proxy, resource interaction

An interesting part of this infrastructure is that the authentication phase is entirely separated by the authorization part.

Using SSL *authentication* protocol the client proves the ownership of the certificate presented (i.e. proves that it owns the private key associated with public key contained in certificate).

Different checks are performed by the *authorization* modules that conform with resource's policies.

The steps performed by resource when connecting to proxy are:

1. connect securely to proxy and authenticates it (it checks if proxy is trusted)

2. delegates his authorization policies (information on how a user must be checked before permitting access);

3. Maintain dynamically this policies;

The steps performed by client are:

1. Authenticate at proxy using a X509 Certificate. The SSL Authentication Protocol (SAP) is widely used for this purpose. Upon the final of this step the client proved his identity and it enters the authorization phase.

2. Based on client identity (X509 Certificate) and resource authorization policy the proxy call different authorization modules that implement specific functionality and decide if client can use the resource.

The proxy service act in this case as a Policy Decision Point and a Policy Enforcement Point (gatekeeper) for resources connected to it. The advantage is that the proxy may act at the same time as gatekeeper not only for a single resource but for multiples and thus, users can request access for a set of resources in a single session.

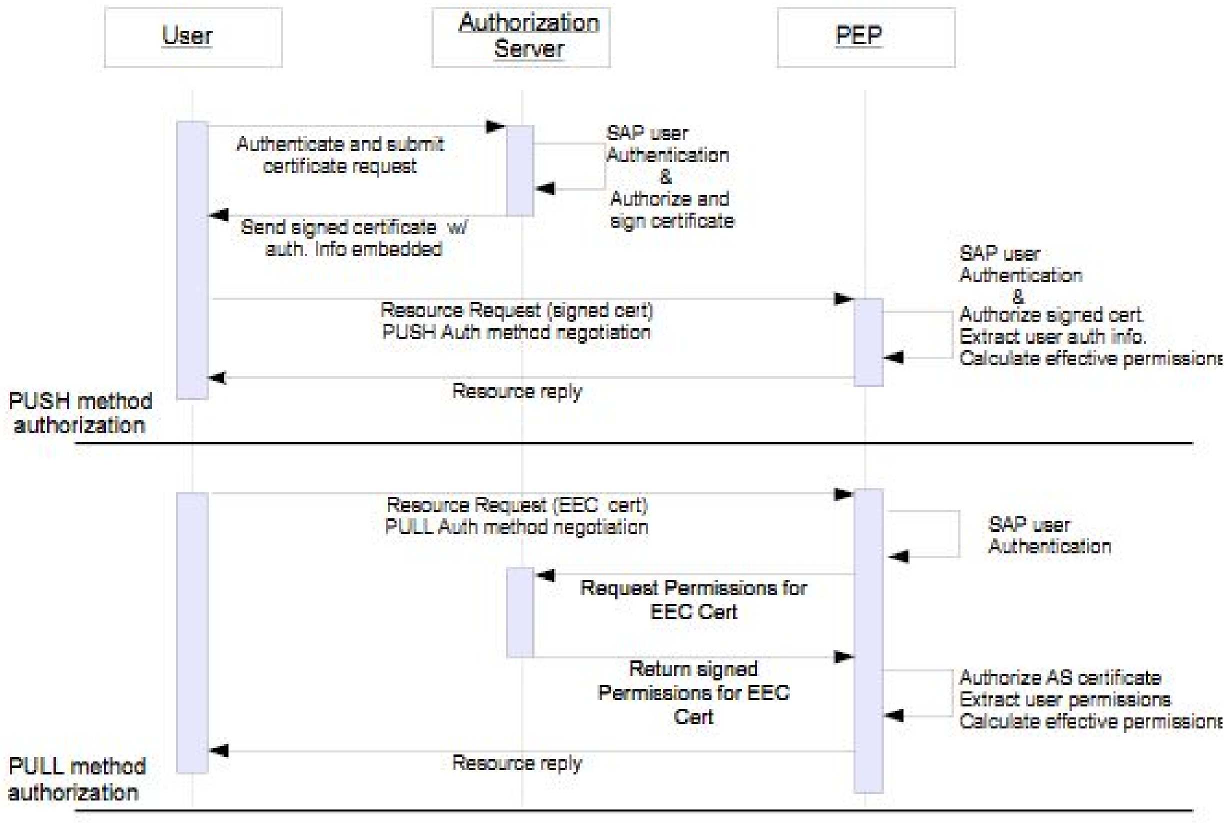
Different pluggable types of authorization can be implemented as authorization modules:

- push model:
- pull model
- local or remote verifying of client credentials (certificates)
- etc.

We will discuss this models in the next sections.

a) ***Push and pull models***

Sequence diagrams:

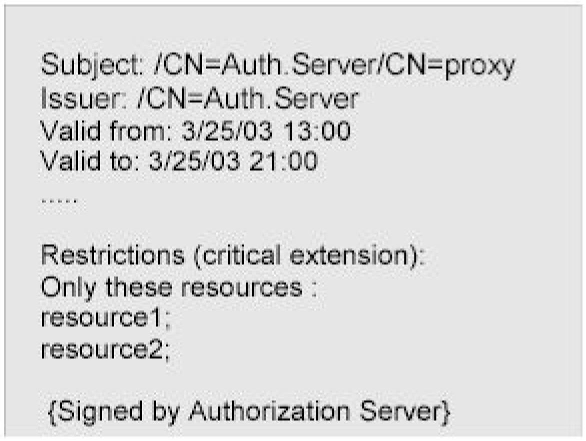

User
Authorization Server
PEP
Authenticate and submit certificate request
SAP user Authentication & Authorize and sign certificate
Send signed certificate w/ auth. Info embedded
SAP user Authentication & Authorize signed cert. Extract user auth info. Calculate effective permissions
Resource Request (signed cert) PUSH Auth method negotiation
Resource reply
PUSH method authorization
Resource Request (EEC cert) PULL Auth method negotiation
SAP user Authentication
Request Permissions for EEC Cert
Return signed Permissions for EEC Cert
Authorize AS certificate Extract user permissions Calculate effective permissions
Resource reply
PULL method authorization

This types of authorization are suitable in situations when a large number of user access resources and the resource providers do not mange the permissions for every user. Sequence diagrams for this models are presented in figure

In *push model* the resource providers specify course-grained access control policies in terms of communities as a whole, delegating fine-grained access control policy management to the user communities who runs a specialized service to authenticate users.

In this way trust relationships are reduced from N*M to C*M (where N is the number of users, C is number of users communities and M is the number of resources).

When a user wants to access a resource, that user makes a request to the authorization server. If the authorization server decide that the user has the appropriate privileges, it issues the user a restricted certificate with an embedded policy giving the user the right to perform the requested actions.


```
Subject: /CN=Auth.Server/CN=proxy
Issuer: /CN=Auth.Server
Valid from: 3/25/03 13:00
Valid to: 3/25/03 21:00
.....

Restrictions (critical extension):
Only these resources :
resource1;
resource2;

 {Signed by Authorization Server}
```

Such a proxy certificate looks like:

As we can see, this certificate has the subject name and issuer of authorization server.

The certificate is a proxy one because he contains a critical section (ProxyCert) and a DN suffix ("CN=proxy") which impose that the verification application must understood the critical section and threat it as a proxy.

The user then uses the certificate from the authorization server to connect to the policy enforcement point for resource. PEP use resource authorization policy to determine if resource trusts authorization server that signed the certificate and if the user's privileges embedded in certificate permit resource access.

This model can be easily implemented in an authorization module. The resource specifies in his authorization policy a list of trusted authorization servers and when a user request access to it using a proxy certificate the module verifies that the certificate was issued by a trusted server.

Pull Model

This model is very similar to push model, but in this case the PEP makes itself a connection to an authorization server and asks it whether a named principal is authorized. The authorization server either responds yes or no to the specific question or returns an access control list that is then checked locally. In either case, the integrity of the connection between the authorization server and the party checking the authorization information must be assures.

The user's privileges can be acquired in two modes:

(1) as an Attribute Certificate(s) (AC) from a certificate repository maintained by authorization server;

(2) as a authorization assertion created on fly by the authorization server based on user privileges maintained in database.

(1) An Attribute Certificates is a certificate that binds arbitrary attributes (e.g. role membership information, policy statements, accounting information) to identities. The binding can be accomplished by specifying either a name (e.g. X.500 Distinguished Name), or/and a X.509 public key certificate issuer and serial number, or/and the public key of an entity as the "holder". A single attribute certificate may contain a set of attributes. A validity field contains a time frame (“not before” and “not after”) that bounds the lifetime of an AC.

The issuer of an attribute certificate (AC) is often referred to as the authoritative entity. The issuer is identified in an issuer field and signs the AC digitally. Thus, ACs are self contained and do not need to be protected or stored in secured/trusted directories. A relying party, before accepting an attribute certificate, must verify that the private key used to create the signature is associated with the issuer (e.g. via the issuers X.509 public-key certificate) and that the issuer was authoritative for the attribute.

An textual representation of an attribute certificate is:

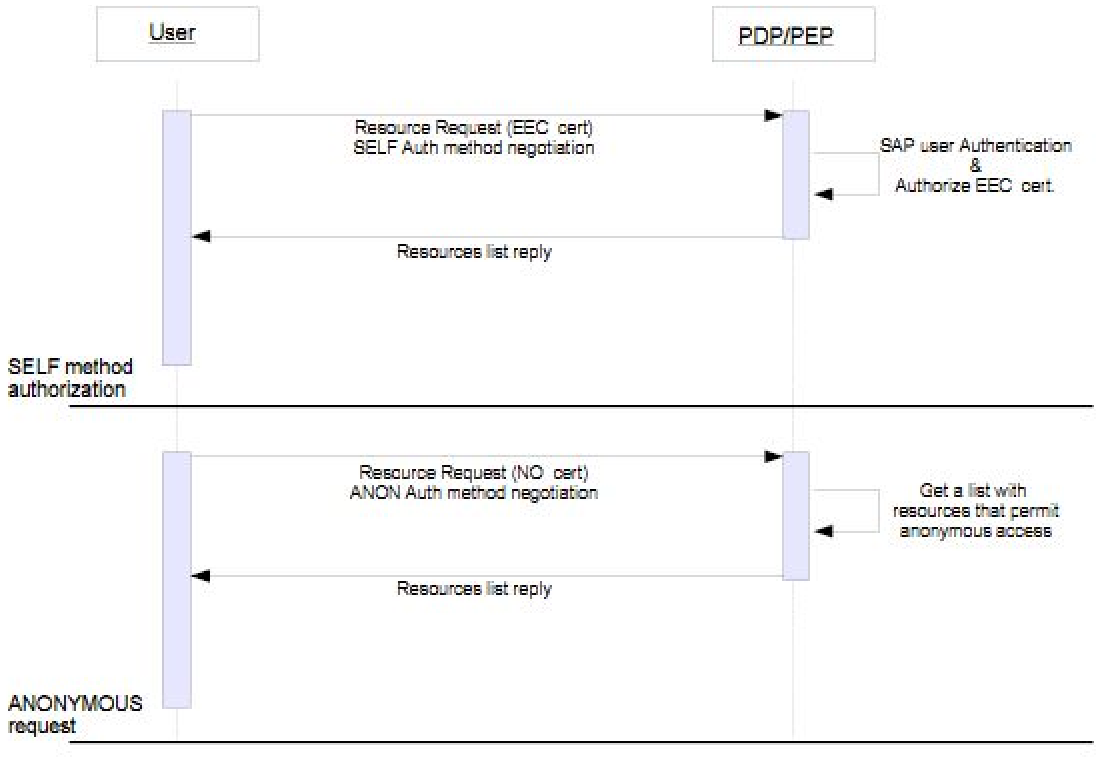
```
[Acinfo:
      Version: 1

      Holder:

            BaseCertificateID: null

            EntityName: [CN=Adi Muraru, OU=RoGrid, O=PUB, C=ro]

            ObjectDigestInfo: {hash of a public key}

      Issuer:

            IssuerName: [/CN=Ramiro Voicu, admin/U=RoGrid/O=PUB/ C=ro]

            BaseCertificateID: null

            ObjectDigestInfo: {hash of a public key}

      Signature: SHA1withRSA
```

```
    SerialNumber: [ 05]

    AttrCertValidityPeriod:  NotBeforeTime:  Sat  May  10  09:27:16  EDT
2004 NotAfterTime: Sat May 30 09:27:16 EDT 2004

    Attributes: 1 [1]:Type: ObjectId: 1.3.6.1.4.1.6760.8.1.1 Values: 1
        Values[0]: <!- an xml encoded attribute->
        IssuerUniqueID: 10110101
        Extensions: null ]
        SignatureAlgorithm: SHA1withRSA
        SignatureValue: <!-- binary data (omitted) -->
```

(2) Another way to obtain a privilege assertion is to use a specialized protocol. The Security Assertion Markup Language (SAML) defines a language and protocol to exchange authentication and authorization information. Its primary goal is to provide a mechanism by which permissions and management data can be shared in a standardized, implementation independent fashion across administrative domains. SAML provides schemas for queries and replies for security related assertions. Attribute assertions for a particular subject may be requested via an AttributeQuery wrapped within a SAML request. According to protocol semantics, a SAML response to that request contains zero or more relevant assertions.

As the assertion is the packaging of asserted data, SAML specifies that digital signatures be attached at this level. However, a single SAML assertion can wrap multiple attribute statements. Each attribute statement contains a single subject identity, and one or more attributes, each with zero or more values.

b) Authorization based on identity certificates and anonymous access.


User
PDP/PEP
Resource Request (EEC cert)
SELF Auth method negotiation
SAP user Authentication
&
Authorize EEC cert.
Resources list reply
SELF method authorization
Resource Request (NO cert)
ANON Auth method negotiation
Get a list with resources that permit anonymous access
Resources list reply
ANONYMOUS request

This type of authorization can be suitable when a resource provider maintain permissions for individual users. The client authenticates itself at PDP/PEP (proxy) using a end-entity certificate. Proxy use resource policy to authorize this certificate (e.g.: checks if it valid,and if it is contained in a truststore of certificates or in a LDAP server indicated by the resource)

But not all resources connected to a security proxy needs authorization. In this case, using the same infrastructure the clients that connect anonymously must access this type of resources.

5.4. Components of the system. Java Implementation

We will present the most interesting parts of the implementation of components of this project:

1. Resource (farm) - authorization policies

2. Proxy (acting as a gatekeeper)

3. A simple authorization server (running in push mode)

4. Client.

1. *Farm authorization policies:*

```
interface FarmAuthPoliciesI extends Serializable {
 public addAuthorizationPolicy(String policyType, Object policyData);
 public getAuthPolicyTypes();
 public getPolicyData(String policyType);
}
```

FarmAuthPoliciesI is a java interface that defines the authorization policies for a particular Farm (policyType define the authorization module that will be activated when the when the policy will be enforced; policyData is an parameter for authorization module).

The interface extends Serializable, since it can be transmitted over a connection.

Authorization policies are defined in a farm configuration file:

```
policyType = push
     #a trustore with authorization servers that may issue proxy certificates
policyData = /path/to/trustedAuthServers.ts

policyType = pull
     #a trustore with authorization servers from where proxy may aquire
attribute #certificates
policyData = /path/to/trustedAuthServers.ts

policyType = local
     #a trustore with users certificates that may access the resource
policyData = /path/to/trustedusers.ts

policyType = LDAP
 #a ldap url where user certificates must be verified
policyData = /url/to/a/ldap-server
```

For a farm that permits anonymous access:

```
policyType = anon
policyData = none
```

The farm also define the proxy that are allowed to do authorization on behalf it:

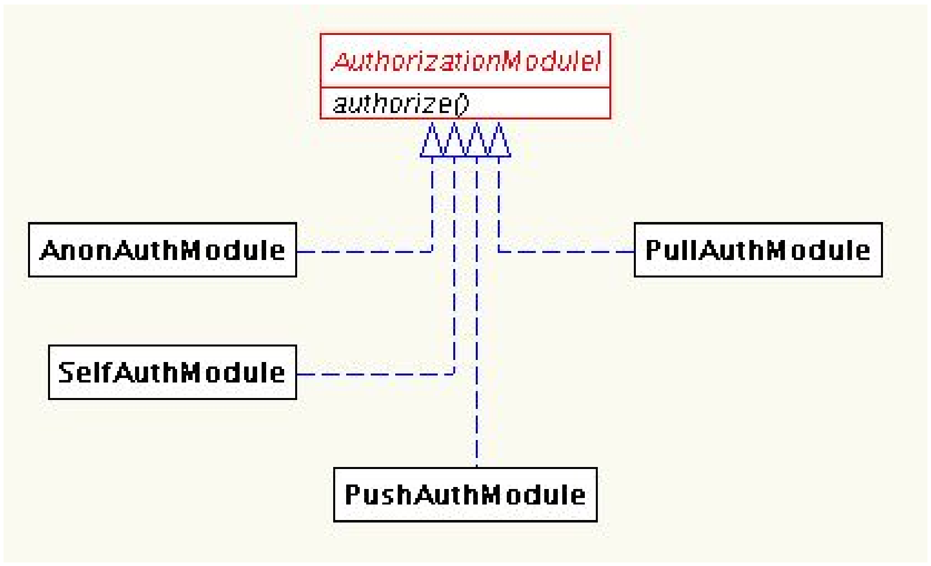
```
trusted_proxies = /path/to/proxy-trustore
```

2. Proxy Gatekeeper

ClientWorker is a Thread class in proxy package that do the communication between proxy and a client.
The important part of implementation (authentication and authorization) in this class is presented in the next paragraph:

```
class ClientWorker extends Thread {

     [....]
          //SAP user authentication
     [....]

     //get authenticated client certificate chain

             SSLSession session = this.socket.getSession();

             X509Certificate[] userCredentials = null;

                try {

                  userCredentials = session.getPeerCertificateChain();

                  } catch (SSLPeerUnverifiedException ue) {

                       //client connects in anonymous mode

                  }

          AuthorizationModuleI am=authfactory.createAuthModule
                                   (userCredentials, farmsPolicies);
          //get a list with farm that permits access to user

          List farms = am.authorize();

     [...]
}
```

Every authorization type is implemented as a class that implements AuthorizationModuleI:



3. A simple *authorization server* running in push mode (clients connects to it and acquire a short-term proxy certificate that and use it later when connecting to a proxy gatekeeper.

Authorization server maintain a repository with authorized clients (a JKS trust-store) and an ACL with permissions mapping for this users. In this simple service we maintain it as a XML file that looks like:

```
<acl>
       <subject>
            <DN>CN=Client1, OU=RoGrid, O=PUB, L=Romania, ST=Romania, C=RO</DN>
            <CA>CN=DummyCA, OU=RoGrid, O=PUB, L=Romania, ST=Romania, C=RO</CA>
           <permissions>
               <resource>farm1_UID</resource>
               <resource>farm2_UID</resource>
               <resource>farm3_UID</resource>
               <resource>farm4_UID</resource>
```

```
          </permissions>
      </subject>
      <subject>
         <DN>CN=Client2, OU=RoGrid, O=PUB, L=Bucharest, ST=Romania, C=RO</DN>
         <CA>CN=DummyCA , OU=RoGrid, O=PUB, L=Bucharest, ST=Romania, C=RO</CA>
          <permissions>
              <resource>farm2_UID</resource>
          </permissions>
      </subject>
</acl>
```

Every request is served by a separate thread that authenticate user and authorize it based on trust-store and acl and return a proxy certificate.

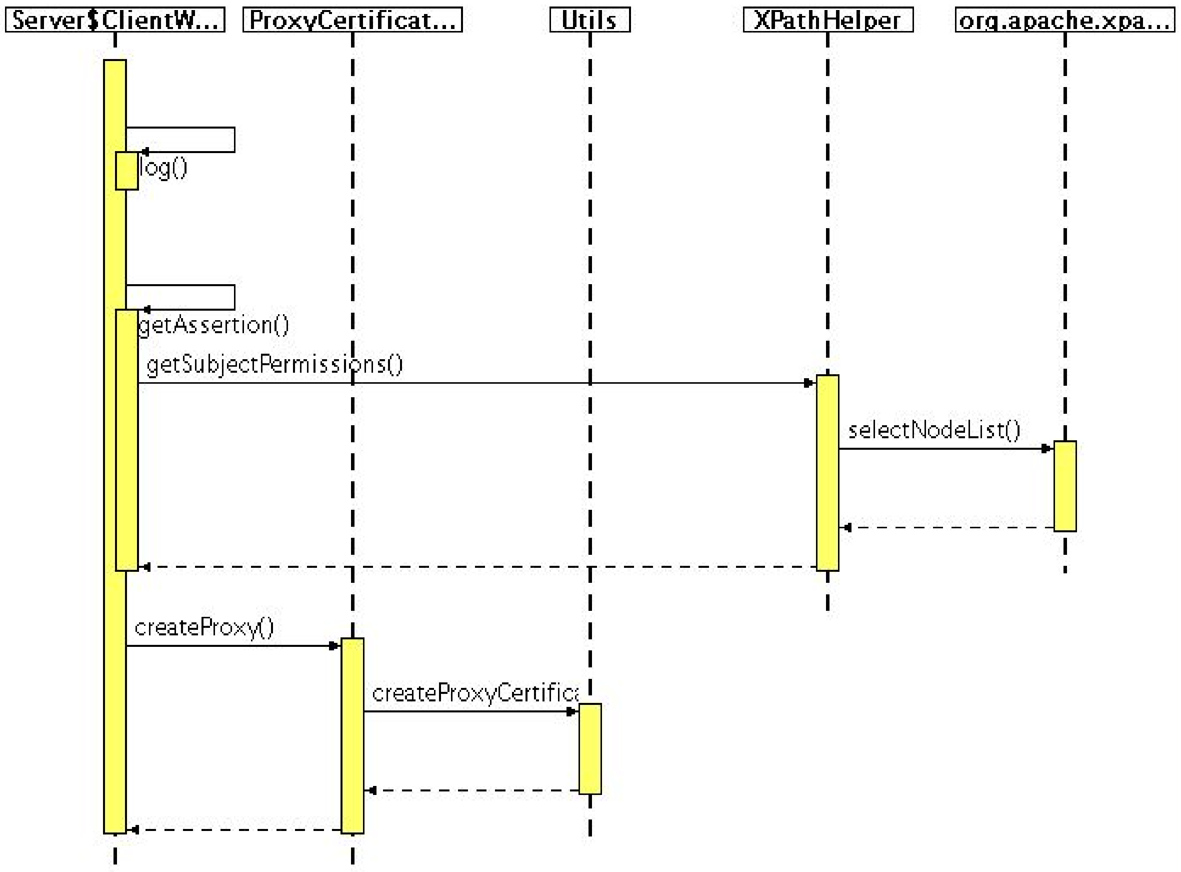
The authentication of client is performed during SSL-Handshake.
Instance of `javax.net.ssl.X509TrustManager` interface manage which X509 certificates may be used to authenticate the remote side of a secure connection. Decisions may be based on trusted certificate authorities, certificate revocation lists, online status checking or other means:

```
     interface javax.net.ssl.X509TrustManager {
      public void checkClientTrusted(X509Certificate[] arg0, String arg1)
            throws CertificateException;
      public void checkClientTrusted(X509Certificate[] arg0, String arg1)
            throws CertificateException;
      public X509Certificate[] getAcceptedIssuers();

}
```

The authorization server implement a such interface and defines the constraints that a user certificate must satisify:

```
class ExtendedTrustManager implements X509TrustManager {

            [....]
            public void checkClientTrusted(X509Certificate[] arg0, String
                                           arg1) throws CertificateException {

                  //verify if is valid

                  arg0[0].checkValidity();

                  //verify  if cert is in trustore
                  String alias=null;
                  //if cert does not exist in trustore or
                  //cert exist and is a keyentry throw Exception

                  if ((alias=ks.getCertificateAlias(arg0[0])) == null ||
                        ((alias=ks.getCertificateAlias(arg0[0])) != null &&
                         ks.isKeyEntry(alias)))

                  throw new CertificateException("Certificate invalid");
            }
}
```

The creation of the proxy certificate is presented in the following diagram:



An important method in this diagram is *createProxyCertificate.*

The authorization server create on-fly proxy certificates for authorized users. This is a form of delegation, the authorization server sign user public-key but impose some constraints on it: the time interval in which this certificate is valid is shortly and a critical extension (with user privileges) is added (the enforcement point must understood this extension, otherwise it cannot validate this certificate).

4. Client.

A simple client has been implemented. It connects to proxy either with with a delegated proxy certificate (requested from AS) (push mode), or with using his own identity certificate (self or pull) or it can make a anonymous request.

The client is started with the following commands:

```
$ cd MSRC/MonaLisa/Security
$ ./authClient
```

authClient is a simple bash script with the following contents:

```
export CLASSPATH="/root/cas/ogsa/impl/java/lib/cog-jglobus.jar:/home/MSRC/bin/"
java lia.SecurityTest.AuthClient.Main \
      -ASHost 127.0.0.1  \
      -proxyHost 127.0.0.1 \
      -at push
```

Prior of starting the client, an authorization server, a proxy and several testing farms was started.

The output of client is:

```
Connecting to MAS ....
Successfull authentication at MAS
Waiting for a MAS delegated certificate....
ProxyCert SubjectDN:CN=proxy, C=RO, ST=Romania, L=Bucharest, O=Pub, OU=RoGrid,
CN=MAS Server
ProxyCert IssuerDN: C=RO, ST=Romania, L=Bucharest, O=Pub, OU=RoGrid, CN=MAS
Server
ProxyCert Validity:Sun Jun 20 21:00:26 EEST 2004:Sun Jun 20 23:00:26 EEST 2004
Connecting to proxy ...
Proxy authorized you. [1] farm access:
SID111114
```

Conclusions

The creation and deployment of computational grids will have a profound impact on the security of distributed systems. Because grid resources are managed by many organizations, often with different security requirements and possibly conflicting security policies managing security for such a system is difficult. Implementing an adaptive security infrastructure based on open-standards seems to be a solution.

MonALISA is an example of such a system. MonaLISA is a robust monitoring system, providing upper layers a flexible framework, allowing rapid development of complex clients, ranging from pseudo-clients that store results in a database, to GUI clients started from a web page. MonALISA is currently monitoring several Grids and distributed applications on around 150 sites.

The security infrastructure proposed eliminates the problems with single-points of failure by distributing the security gateways and it assures a high level of availability and scalability. Currently we evaluate this security infrastructure in MonaLISA framework.

We think that the dynamic and flexible security infrastructure proposed (with a modularized security gateway acting as a gatekeeper for resources) should be suitable for many types of architectures.

An item for future work is to adapt this infrastructure for web-services architecture.

Glossary

Access Control Policy
See Policy.

- *Attribute*
  A named statement of a property (i.e. type value pair).
  Example: a characteristic of a person or other identifiable entity.
- *Attribute Certificate, X.509 Attribute Certificate*
  A Certificate (see Certificate) that binds attributes to entities. X.509 Attribute Certificates follow a standardized structure, are ASN.1 encoded and use public- key cryptography to digitally sign the certificate structure.

  An attribute certificate allows a user attribute certifier to provide a user characteristic in a natural and convenient way.
- *Authenticate*
  To verify the identity of another party in a communication.
- *CA*
  Abbreviation for certificate authority.
- *Certification authority*
  An entity trusted to "vouch" for the identity of a subject. In a public key infrastructure, a certificate authority signs an identity certificate for the subject
- *Certificate*
  A digitally signed document used in public key cryptography that associates either a public-key with an entity (see Identity Certificate) or an attribute with an entity (see Attribute Certificate) Certificates are issued by authorities and have a lifetime associated. Relying parties that accept certificates must trust the certificate issuer to be authoritative for the key or attribute to identity binding made in the certificate.
- *Credential*
  A security token, often consisting of a public component (e.g. a certificate and public key) and a private component (the corresponding private key).

  Distinguished name
  The identifier associated with an entity (e.g., a person) in the ISO X.500 Directory. The distinguished name's format is not defined in the LDAP specification(see the references section for a link to the current protocol

specification), but conventionally it is a representation of the entity's position in a hierarchy, such as that formed by a person's country, organization, and organizational unit, together with the person's common name.
Abbreviation: DN.

- *CN*
  Abbreviation for common name.
- *Common name*
  A person's given name, e.g., Mary R. Thompson.
  See also distinguished name.
- *Delegation*
  An Entity grants the ability to act on its behalf to another entity.
- *Digital signature*
  A cryptographic procedure that allows one entity to verify that a particular piece of data was produced by a particular subject.
- *DN*
  Abbreviation for distinguished name.
- *End-entity Certificate*
  See Identity Certificate.
- *Fine-grained / Fine-grained access right*
  Fine-grained is a relative quality that is interpreted with respect to the resources and applications of interest. Ex: the privilege: user X is allowed to read file F is considered fine-grained while the privilege user X is 161 allowed to exercise any right assigned to account Y is not considered fine-grained. A coarse-grained privilege can, of course, be expressed in a fine-grained system. In other contexts, fine-grained may refer to even smaller entities than those considered in this work.
- *Identity Certificate, Public-key Certificate, End-entity Certificate*
  A certificate binding an identity to a public key. X.509 defines a profile in ASN.1 for identity certificates. Identity Certificates typically have lifetimes on the order of one year and can be renewed.
- *LDAP*
  Abbreviation for the Lightweight Directory Access Protocol
- *Lightweight Directory Access Protocol*
  A protocol "designed to provide access to the X.500 Directory while not incurring

the resource requirements of the Directory Access Protocol" [RFC 2559].
The Lightweight Directory Access Protocol (LDAP) is used to communicate with the ISO/OSI directory service. Broadly defined, a directory is a "special purpose [database], usually containing typed information. " An example of an Internet-based directory is the Domain Name Service (DNS). A directory accessed via LDAP, however, can contain any kind of information, unlike the special-purpose DNS directory. We refer to a directory accessible via LDAP as an LDAP server.
An LDAP server is used as a Registration Agent (RA) by the Netscape CA. All valid certificates are entered into an associated LDAP server, and are removed when then are revoked. Thus one can check if a certificate has been revoked by looking it up in the CA's LDAP server. If it is not found, it is assumed to have been revoked.

- *Message integrity*
  Protection of communication, which ensures that the contents of a message cannot be modified by an attacker.
- *Message confidentiality*:
  Protection of communication, which ensures that the contents of a message cannot be read by an attacker.
- *Non-repudiation*: A cryptographic procedure that allows one entity to prove that a piece of data was undeniably produced by a particular subject, perhaps at a particular time.

- *Policy*
  A machine readable representation of a set of rules that specify the intent of the security model. Access Control Policies specify access control rules that apply to a set of resources.
- *Policy Decision Point (PDP)*
  A component of an authorization architecture that makes authorization decision upon request based on attributes and policies. PDPs are typically application independent and do not understand application semantics and protocols.
- *Policy Enforcement Point (PEP)*
  A component of an authorization architecture that enforces authorization decisions made by a PDP. PEPs are often integrated with the application.

- *Proxy Certificate*
  A Proxy Certificate is a certificate with a short lifetime (e.g. on the order of one day) that binds a temporary public key (similar to a session key) to a unique proxy identity that is derived from a subject identity. Proxy certificates are issued by subjects and signed with the private key corresponding to the subjects Identity Certificate (the subject s long term private key). Proxy certificates can be used to authenticate with other grid entities on the issuer s (subject s) behalf without requiring access to the subject s long term private key.
- *Public-Key Certificate*
  See Identity Certificate.
- *Resource Attributes*
  Attributes that describe the characteristics of resources. For example the clearance level a resource is certified under.
- *Relying Party*
  An entity that makes decisions based on identity and attribute certificates.
- *Subject Attributes*
  Attributes that describe the characteristics of subjects. For example the membership of a subject in a group of subjects.
- *Secure Sockets Layer protocol*
  A network protocol that allows the two ends of a unicast communication link to authenticate one another and to establish an encrypted connection.
  Abbreviation: SSL.
  See also Transport Layer Security protocol.
- *SSL*
  Abbreviation for the Secure Sockets Layer protocol.
- *TLS*
  Abbreviation for the Transport Layer Security protocol.
- *Transport Layer Security protocol*
  The IETF's adaptation of SSL, version 3. The IETF's Transport Layer Security working group is in charge of the standardization process.
  Abbreviation: TLS.
- *Validity*
  Validity in the context of certificates typically refers to the certificate lifetime and its integrity. A relying party must verify that the certificate lifetime includes the

time at which it is being used and that the signature is authentic by verifying it with the issuer s public key.

- *X.509*
  The ISO authentication framework.

Bibliography

1. B. Clifford Neuman - Security, Accounting, and Assurance chapter in The GRID: Blueprint for a New Computing Infrastructure

2. Kerberos.General Informations http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html

3. R. Housley, W. Polk, W. Ford, D. Solo, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile , Internet RFC3280, Internet Engineering Task Force, Public Key Infrastructure Working Group, April 2002

4. X.509 Proxy Certificates for Dynamic Delegation. V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. 3rd Annual PKI R&D Workshop, 2004.

5. Security for Grid Services. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, to appear June 2003.

6. A Community Authorization Service for Group Collaboration. L. Pearlman, V. Welch, I. Foster, C. Kesselman, S. Tuecke. Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

7. A Security Architecture for Computational Grids. I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998.Describes techniques for authentication in wide area computing environments.

8. Fine-Grain Authorization for Resource Management in the Grid Environment. K. Keahey, V. Welch. Proceedings of Grid2002 Workshop, 2002.

9. Privilege Management and Authorization in Grid Computing Environments
Markus Lorch

10. A National-Scale Authentication Infrastructure. R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, V. Welch. IEEE Computer, 33(12):60-66, 2000.

11. MonALISA: A Distributed Monitoring Service Architecture
H.B. Newman, I.C. Legrand, P.Galvez, R. Voicu, C. Cirstoiu
CHEP 2003, La Jola, California, March 2003

12. A Distributed Monitoring System: MonALISA
R. Voicu
CSCS Conference, Bucharest, July 2003

13. MonALISA web page http://monalisa.carc.caltech.edu