Sygn: A certificate based access control in Grid environments [⋆]

Ludwig Seitz, Jean-Marc Pierson, and Lionel Brunie

LIRIS, CNRS UMR 5205, INSA de Lyon
7, av. Jean Capelle, 69621 Villeurbanne cedex, FRANCE
{ludwig.seitz, jean-marc.pierson, lionel.brunie}@liris.cnrs.fr

Abstract. In this paper we study the problem of Grid access control in environments with high confidentiality requirements and a large number of users. We propose a novel access control mechanism, *Sygn*, that implements decentralized permission storage and management. All permissions in *Sygn* are encoded in certificates, which are stored by their owners and used when required. *Sygn* allows for decentralized administration of dynamically changing resources and permissions. *Sygn* also supports role based access control. Prmissions can be created on demand without the need to contact a central permission storage system. The *Sygn* access control servers store only minimal security critical information to minimize the impact of a successful attack. Sygn has been successfully integrated in a lightweight grid middleware.

Keywords: role based access control, distributed access control, Grid access control, certificate based access control

1 Introduction

Applications such as particle physics, terrestrial observation or biomedical research require lots of computing power and disk storage space. Users working on such applications often face the problem that their organization does provide them with the resources required for processing their data. Moreover projects to share resources between different partners are often confronted with the fact that heterogeneous resources require great effort to be used together. Computational Grids [1] strive to solve these problems by providing a middleware that transparently manages the various resources (data, storage capacity and processing power), hiding the heterogeneity to the end-user.

Grids have been identified as a possible approach for *health-care networks*. Indeed in health-care increasing amounts of computerized patient data is produced and used. This data requires storage and processing. Furthermore, availability of medical data stored at remote sites can be crucial for successful treatment of a patient.

Due to legal liability such networks will be of no practical use without privacy protection of the medical data, as they will not be accepted by their intended users, as long as this problem has not been dealt with.

[⋆] This work is partly supported by the Région Rhône-Alpes project RAGTIME and the French ministry of research ACI-GRID MEDIGRID project

A part of this protection has to be provided by a data access control mechanism. Members of the medical staff have different access rights to the data of the patients they are treating. These rights are typically specific to the functions they perform (doctor, nurse, secretaries etc) and not to the person herself. Some of the functions have a hierarchical permission structure (e.g. the rights of a nurse are a subset of the rights of a doctor). Consequently role based access control (RBAC)[2] is ideally suited to manage this kind of permission structure.

Classical RBAC mechanisms often do not provide data access control at a fine-grained level. However since the sources of authority for data are legally responsible for the privacy of the data they control, they should be in charge of access control permissions to their fine-grained data.

The Grid environment creates additional challenges that have not been addressed by classical RBAC mechanisms. Grids assemble heterogeneous resources from different providers. The physical storage location of data is transparent to the users and data may be automatically replicated by the Grid infrastructure for a more effective access. Therefore a user who stores sensitive data on a Grid wants the access control to be effective for all its replicas, no matter who owns the actual storage devices.

To allow users to selectively share data using the Grid, and to enable proxying the access control system must allow them to delegate access rights to other entities. The possibility to delegate access rights also makes the system more scalable, since it allows to distribute the burden of access control administration.

In Grids, services have a dynamic availability. This means that a service (e.g. a storage device) may spontaneously connect to and disconnect from the Grid. This can cause consistency problems in access rights of replicated data. Furthermore precautions must be taken to deal with the case of the access control service itself becoming unavailable.

Administrators of storage and computing resources want to be able to control the access to the resources they offer, or they will be reluctant to provide them on the Grid.

In the case of the medical Grid, it is also necessary to be able to grant ad-hoc permissions to access data. As these permissions can be needed in emergency situations, they should not involve any time consuming administration procedures.

Finally when dealing with medical data it is a legal requirement to log all accesses to these data, so that misuse can be traced and liabilities can be assigned.

We can therefore summarize the requirements for an access control system in a medical Grid environment as follows:

- It must support role based access control.
- It must permit fine-grained access control on data, where the sources of authority (SOA) that may issue permissions are individual users owning the data.
- It must support delegation of access rights.
- It must keep access rights consistent for all data including all replicas of some piece of data. Rights must remain consistent even if a change occurs while replicas are unavailable.
- It must handle outages of the access control service itself gracefully.
- It must enable administrators of disk space and computing power to control access to their resources.

– It should support ad-hoc permission granting.
– It must provide the traceability of all accesses to data.

MEDIGRID [1] is a project funded by the french ministry of research. Its objective is to investigate the use of Grid infrastructures for health-care networks. Within this project one topic of research is the security infrastructure for medical Grids. The creation of the access control service presented here has been a part of this work.

The rest of this publication is organized as follows. We present current access control systems in section 2. We then argue why these systems do not fulfill the requirements for a medical Grid access control. In section 3 we propose *Sygn*, an access control system designed according to above requirements. Then we discuss the *Sygn* approach in section 4 and finally conclude and present future work in section 5.

2 Related Work

The Community Authorization Service (CAS)[3] from the Globus Alliance and the Virtual Organization Membership Service (VOMS)[4] provide conceptually similar access control services for Grids.

In both approaches Virtual Organizations (VOs) are granted bulk rights to various Grid resources. Each VO uses a central server that issues subsets of the VO's rights to the VO members. Whereas the CAS server stores all authorizations concerning the VO members, VOMS only stores the group/role memberships and the local resources are expected to maintain access control lists that attribute various access rights to these groups/roles.

The fact that both services centralize unprotected access control information makes them trusted third parties and potential bottlenecks. They can therefore become single points of failure, e.g. if an attacker compromises a CAS server he has access to all resources granted to the community that this CAS manages. Furthermore the fact that both services are centrally managed and that resources grant bulk rights to the VOs make fine grained data access control and ad-hoc granting of rights very difficult to manage.

Akenti [5] extends the VO model of CAS and VOMS by providing a way to express and enforce access control policies without requiring a central service that administrates them. Aktenti uses signed certificates to store authorization information. This protects the information against unauthorized modification and makes it possible to store it on less secured sites. Currently Akenti uses a proprietary authorization certificate format.

Similar to Akenti, the PERMIS [6] authorization infrastructure uses certificates to securely store permissions. In contrast to Akenti PERMIS relies on the X.509 attribute certificates (AC) [7] as authorization certificate format. The X.509 AC specification has some serious limits, since it discourages the use of certificate paths, which are considered too complex to administrate and process. Therefore delegation using X.509 AC paths is not supported. Furthermore, the specification states that, for each particular set of attributes, only one source of authority (SOA) may exist. These limitations make the

[1] http://creatis-www.insa-lyon.fr/MEDIGRID

use of X.509 ACs questionable with respect to distributed access control of dynamic resources.

Shibboleth[8] is an access control solution for web services, that uses the permission pull approach like Akenti and PERMIS. Our concern with Shibboleth is that its design assumes that the source of authority for authorizations concerning a specific user is always his home organization. Such an assumption is not necessarily true in Grids, where users may be granted different permissions from multiple sources of authority.

A further general concern with Akenti, PERMIS and Shibboleth is that their *Policy Decision Points* (PDP) use the *pull* approach [9] to retrieve the necessary information for authorization decisions . In the pull approach, the PDP retrieves (i.e. *pulls*) the necessary authorization information itself, upon receiving an access request.

It is therefore difficult for the user to enforce the principle of using the least privilege for each operation. Badly configured or malicious PDPs may pull more authorization information than necessary for a request and the user would be incapable to notice this. Shibboleth handles this problem by introducing permission release policies that can be configured by the users. However we deem this approach to be more cumbersome and less intuitive to handle than just selecting the permission you want to use for a specific request.

Another drawback of the pull model is that it puts the load of retrieving the authorization information upon the PDP and does not allow to temporally decouple the authorization assertion from the actual request. We therefore think that the *push* approach, where the request issuer provides the necessary permissions to the PDP is more appropriate for handling ad-hoc permissions in an environment with dynamically changing resources.

Cardea [10] is an access control solution for distributed systems. It is based on the standard policy language XACML [11] and the standard assertion language SAML [12] to certify authorization information. Since XACML is based on the pull model, the same concerns as for Akenti and PERMIS apply. Furthermore neither the current SAML nor the current XACML standard consider delegation of access rights. Due to this limitations we have the same concerns as for X.509 ACs with regard to XACML and SAML based systems.

PRIMA [13, 14] is a Grid access control system that specifically supports ad-hoc permission granting. PRIMA is a hybrid push/pull architecture, where attributes are pushed to the PDP and general policies are pulled by the PDP. PRIMA uses XACML as policy language and X.509 ACs for authorization. However the designers of PRIMA have created a workaround to enable delegation within those standards. PRIMA maps the data access permissions of a user to local POSIX.1E file system access control lists or Grid Access Control Lists[15]. This approach makes it more difficult to realize the Grid paradigm of integrating heterogeneous systems, since it requires one of those specific systems to be deployed at operating system level on all hosts participating in the Grid architecture.

SPKI[16] is an IETF proposal for a simple public key infrastructure focused on authorization. It introduces a simple format for authorization certificates. An access control list (ACL) that is co-located with each resource specifies the public keys of the sources of authority (SOAs) for this resource. These SOAs may issue permissions on the resource and authorize other users to delegate them. SPKI uses public keys as a means to identify entities and to create unique namespace identifiers. Furthermore it specifies a delegation mechanism through chains of certificates and details tuple reduction rules to produce an authorization decision out of such a certificate chain. Work on SPKI standardization has ceased since 2001 and thus important questions such as implementation of standard RBAC using SPKI have not been addressed. Our proposal takes up several ideas of SPKI as described in 3.

None of the presented systems specifically addresses our requirements of traceability and consistency of access rights for replicated data.

3 System design

This section presents the design of Sygn[2]. It is subdivided as follows: Subsection 3.1 presents the fundamental design approaches of Sygn. Subsection 3.2 explains the structure of a Sygn permission certificate. The semantics of a permission certificate are explained in subsection 3.3. In subsection 3.4 we present how a Sygn server is set up. Finally subsection 3.5 describes how the system processes a request to generate an access decision.

3.1 Fundamental design approaches

Sygn supports the concept of *roles* based on the NIST standard for RBAC [17]. A role is a collection of permissions, that are required to perform a task. Roles are assigned to entities that may activate them selectively to make use of the role permissions. Sygn leaves the semantics of a role (i.e. the task it allows to execute) to the discretion of the role SOA. Any user can create new roles and to select the SOA for this new role in Sygn.

An entity that is SOA for a specific permission can assign it to any existing role without intervention of the role's SOA.

When a role is created in Sygn it must specify a repository, where the permissions assigned to the role are stored for review purposes. These repositories are used to support the review functions required by standard RBAC.

Sygn also supports the concept of hierarchical roles. A role may be assigned to be hierarchically inferior to some other role by its SOA. This means that the superior role inherits all permissions given to the inferior role.

To permit a separation of duties, Sygn allows to specify restrictions on each role-to-user assignment. These restrictions specify other roles that may not be activated in the same request together with this role. As the verification of these restrictions is done during the access request, this corresponds to what the RBAC standard calls *Dynamic Separation of Duties*.

[2] Sygn is a name that comes from the Nordic mythology. It designs a goddess of truthfulness but also of doors and locks

In order to allow an individual fine-grained access control, Sygn uses authorization certificates (AC) for permission granting and delegation. SOAs can thus grant access to their resources by issuing digitally signed ACs.

To allow permission consistency in presence of replicas we have opted for a *permission push model*, where the users store all their permissions themselves. This allows users to use the same permissions on any replica of a file, provided that the replica identifier is the same as the one of the original file. Furthermore, this architecture supports ad-hoc granting of permissions. SOAs can directly give permissions to users, without intervention of any third-party permission storage system. However the push model requires a permission revokation scheme. Thus every Sygn access control server maintains a list of revoked ACs. These lists are updated regularly by consulting redundant certificate revocation services.

To prevent failures of the access control due to service outages, our design deploys the access control services locally on the resource sites. Therefore the access control for a site will only be unavailable when the site itself is also offline.

Sygn allows local administrators of resource sites providing storage space and computational capacity to control the access to their resources. This requires some externally measured metric that evaluates the resource usage. The Sygn system also allows local administrators to set up blacklists of entities that may not access their resources, regardless of their permissions.

To provide traceability, Sygn can be configured to create non-repudiable logs of all access requests. In this configuration, users are required to digitally sign all the requests they submit.

Since our system is intended to be a scientific proof of concept we have decided not to implement standard policy languages such as XACML [11] or standard certificate formats such as SAML [12] or X.509 Attribute Certificates [7]. The justification for this choice is that these standards make the permissions hard to create, manage and read. In X.509 this is due to the ASN.1 encoding of the certificates. XACML and SAML are based on XML, provide a lot of functions and are very generic. This makes them very verbose and adds a lot of overhead to the most simple permissions. Our current proprietary permission encoding is based on XML and could be replaced by a standardized encoding format without modifying the general functionality of Sygn.

Figure 1 shows the deployment and interaction of Sygn modules. In a first step a resource's SOA installs or stores a resource on a (possibly remote) resource server. Using his administration client, he contacts the *resource server* and registers himself as source of authority for this resource in the metadata-base of the local *Sygn policy decision point* (PDP) . For each resource, the Sygn PDP stores only its SOA identifier.

To grant access to a resource consumer (step 2) the SOA issues ACs that allow access to the resource. Note that this process involves only the SOA and the consumer(s), and can be done offline.

The user stores and retrieves ACs as needed. To access the resource, the user contacts the corresponding resource server (we assume that the localisation of the correct server is realised by the Grid middleware) and submits the request that contains the required ACs as shown in step 3.

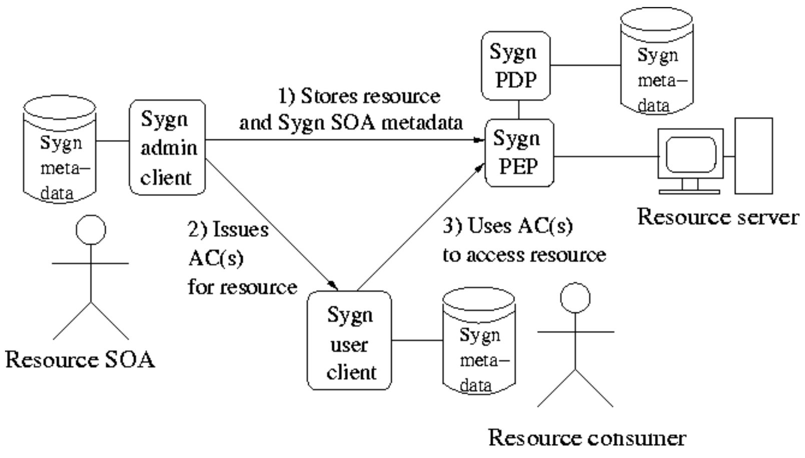


Fig. 1. Deployment of, and interaction between Sygn modules on a Grid.

The resource server needs two different Sygn modules. The *Sygn Policy Decision Point* (PDP) that produces an access control decision based on the *Sygn request* provided by the user and a *Sygn Policy Enforcement Point* (PEP). The PEP interfaces the PDP with the Grid middleware and thus makes the PDP independent of specific Grid implementations. The PEP has two functions: First it has to make sure that the Sygn request corresponds to the actions the user tries to perform on the Grid (otherwise a user could submit valid ACs for one resource together with commands initiating Grid actions that concern another one). Second it uses the Grid authentication mechanisms to make sure the request issuer is the owner of the accompanying ACs.

The Sygn request submitted by the user to the server contains the target resource the user wants to use and ACs authorizing access to this resource. Based on this information, the Sygn PDP decides if the Sygn request is correct and if the ACs correctly authorize this request. Sygn traceability is configured at the PDP level. If turned on, all requests are logged in the Sygn metadata-base. Non-repudiation of those logs can be achieved by another PDP configuration option, that requires timestamping and digital signature of requests by the issuer.

3.2 Structure of a Sygn permission

We chose XML to represent the data structures used in a Sygn AC. The reason for this is that XML is human readable (as opposed to binary encodings such as ASN.1) which makes creation and understanding of certificates in this format easier for the users that have to deal with them. The use of XML is further facilitated by the fact that a large variety of tools supporting the use of XML exist.

Basically a Sygn permission consists of:

– an *identifier* generated by calculating the SHA1-hash of the other AC data.
– an AC *creator*,
– an AC *owner* (also called the *subject* of the permission),

– a *capability* that is given to the owner by the creator, consisting of an *object* and an *action*,
– *validity period limits* in the form of two *timestamps*.
– *restrictions* on the use of the permission,
– a *delegation* depth limit and
– an *electronic signature* that makes unauthorized modifications of the permission detectable[3]

Figure 2 shows an example Sygn permission certificate encoded in XML. Please note that the creator of this AC is not the SOA of the file it authorized access to. Therefore to be usable in a certificate path, other ACs must be present in which the file's SOA allows the creator of this AC to delegate read permissions on the file.

```
00 <AUTHORIZATION_CERTIFICATE>
01    <ID>bA1lxGTYDd3eHT/gr/6B1N4dCWU=</ID>
02    <CREATOR><USER_ID>MIG...aQIBEQ==</USER_ID></CREATOR>
03    <OWNER>  <USER_ID>MIG...wdUQIBEQ==</USER_ID></OWNER>
04    <CAPABILITY>
05       <CAPABILITIY_ID>8k9AiT4a...br8U=</CAPABILITIY_ID>
06       <OBJECT><UNIQUE_FILE_ID>
07          <FILE_SOA>
08             <USER_ID>MIGdMA0...j4jQIBEQ==</USER_ID>
09          </FILE_SOA>
10          <LOGICAL_FILENAME>+/A...88=</LOGICAL_FILENAME>
11       </UNIQUE_FILE_ID></OBJECT>
12       <ACTION>read</ACTION>
13    </CAPABILITY>
14    <NOT_BEFORE>2003-10-01T10:23:02Z</NOT_BEFORE>
15    <NOT_AFTER>2004-10-01T12:22:03Z</NOT_AFTER>
16    <NOT_WITH>...</NOT_WITH>
17    <DELEGATIONS>1</DELEGATIONS>
18    <SIGNATURE>qcIiRan3kpDEPi...eOtc1OV5Byw=</SIGNATURE>
19 </AUTHORIZATION_CERTIFICATE>
```

Fig. 2. An example of an AC where the creator (line 02) grants read access (line 12) on a file (lines 06-11) to the owner of the AC (line 03), with the right to delegate this capability one step (line 17).

Creator The creator of a Sygn permission must be a user identified by a public key. The corresponding private key is used for the signature of the certificate. A public key representing a user is called a *user identifier (UID)* in Sygn. Note that the public key is considered sufficient as unique identifier. Sygn does not associate a distinguished name

[3] We currently use the RSA signature algorithm with SHA-1 hashing and PKCS1v15 padding.

to public keys for user identication such as it would be the case in X.509[19] certificates. Given the size of the namespace (valid public/private-keypairs) it is extremely improbable that two users will accidentially be assigned the same Uid. The concept of using public keys to represent user identities follows the *SPKI* approach.

Sygn subjects Subjects in Sygn are users and roles. They are identified by *subject identifiers* (SID). SIDs are used either as owners of an AC or as SOA's of Sygn objects. A Sygn subject can either be a user represented by a UID or a *role* which is represented by a *role identifier (RID)*.

A RID consists of three elements: A *role name*, which can be any string and should describe the role in question. A *role SOA* of UID type, who initially is the only person that can activate the role. This implies that the SOA of a role can always use the role permissions. The third element of each role is a *review repository*, which specifies a repository where all permissions concerning that role should be stored.

Sygn objects Objects in Sygn are always part of a capability. An object can either be a file identifier (FID), a file set identifier (FSID) referring to a collection of files, a role-object identifier (ROID) which is actually a wrapper that allows to use roles as permissions objects, or a resource identifier (RESID) that identifies a specific hardware resource (e.g. some storage device). Sygn is designed to make it easy to add new types of objects.

An object is identified by two elements: An object name and the object's SOA. For file identifiers, file set identifiers and resource identifiers any type of Sygn subject can be used as SOA. Role object identifiers require the SOA to be of UID type.

Sygn actions Actions are identifiers for possible actions with respect to an object. Sygn currently supports *read*, *write*, *add-to-set* and *remove-from-set* for files and file sets, *activate* for roles and *grant* for hardware resources. The grant action has an additional parameter specifying a scalar value measuring the amount of the hardware resource that is granted. New actions can be easily added if required.

Sygn capabilities The capability is a compound structure, consisting of an *object* and an *action*. To be allowed to grant a capability a user must either be the object's SOA or be authorized by a valid delegation chain from the SOA. Each capability gets an identifier that is generated by calculating a SHA1 hash of the object and action data.

A special type of capability allows SOA's of files or file sets to add those to another file set. Master set is specified as secondary object, after the file or set that is added to it, in this special capability type.

Sygn restrictions Sygn permissions support restrictions in form of a list of RID's that may not be used in together with this AC. This makes it possible to enforce Dynamic Separation of Duties (DSD) as specified in the RBAC standard.

Delegation The *delegation* depth limit is an integer value, that specifies how many steps the AC's capability may be delegated. A limit of zero means the capability may not be delegated at all. Any limit greater than zero allow the AC owner to delegate this capability with a delegation depth limit reduced by at least one.

3.3 Semantics of a Sygn permission

The meaning of a Sygn permission is mostly straightforward. If the owner of an AC is a UID then the user corresponding to the UID can use the AC's capability. However some details must be explained for the correct usage of the more powerful functions of Sygn.

If a role is the SOA of an object, any user able to activate that role can act as that object's SOA.

If an AC has a ROID as object, it gives the AC owner the permission to activate that role and use its permissions.

If the owner of an AC is a role this means every user being able to claim activation of this role (through use of another AC) can use the AC's capability.

Combined together, these two semantics implement hierarchical RBAC. This means that if a role A is subject and role B object of an AC, role A can use all permissions of role B. Role A is said to be hierarchically superior to role B.

The following example illustrates this. Consider the following ACs:

– AC_1 permits *read access on* $file_I$ to $role_B$
– AC_2 permits *read access on* $file_K$ to $role_A$
– AC_3 permits *activation of* $role_B$ to $role_A$

Then $role_A$ is said to be hierarchically superior to $role_B$ since users who can only activate $role_B$ can only read $file_I$ while users who can activate $role_A$ can also activate $role_B$ and thus read $file_I$ *and* write $file_K$.

Permissions can grant file actions on file sets. Such permissions are then applicable to any file that is member of the set. In order to use such a set permissions on a specific file, the certificate adding this file to the set has to be presented.

3.4 Sygn server meta-data

In this subsection we describe the necessary steps for the deployment of a Sygn server. The server relies on a database for efficient meta-data storage and access. The meta-data are critical, therefore great care should be taken when setting up the database to avoid introducing a weakness through insecure database access.

Sygn servers are designed to be deployed locally at the resource sites. There Sygn server needs to have the subject-id of each SOA that has some file or hardware resource on this site.

The Sygn servers provide meta-data storage to record hardware resource usage. Hardware resource SOAs can specify a value for the allowable use of their hardware resource by a user. The definition and measurement of this metric must be implemented in the Sygn PEP.

The server meta-data also provides storage for the ids of revoked certificates, together with their expiration date. A Sygn command permits to erase revoked certificates that have expired from the meta-data base. This function can be used to implement a certificate revocation list (CRL) mechanism to allow SOAs to invalidate ACs they issued before expiration.

Finally the Sygn meta-data includes a blacklist for banning UIDs. This function can be used by the Sygn administrator to exclude users who have abused the local systems resources. Requests from a blacklisted UID will inevitably be refused, no matter what other permissions are submitted.

Having set up the required meta-data the last step to make the system work is to set up the Sygn PEP that matches requests for Grid operations against the requested Sygn permissions and interprets the responses of the Sygn PDP.

3.5 Access decision process

As mentioned before, Sygn is a push architecture. The ACs supporting a request must be sent to the Sygn server together with the request. The structure containing a chain of ACs supporting one request is called a *path*. A path grants access to a resource called the path's *target*. The certificates in a path must trace a valid connection from the SOA of the target to the user who issued the request.

To be valid, a path must satisfy multiple conditions:

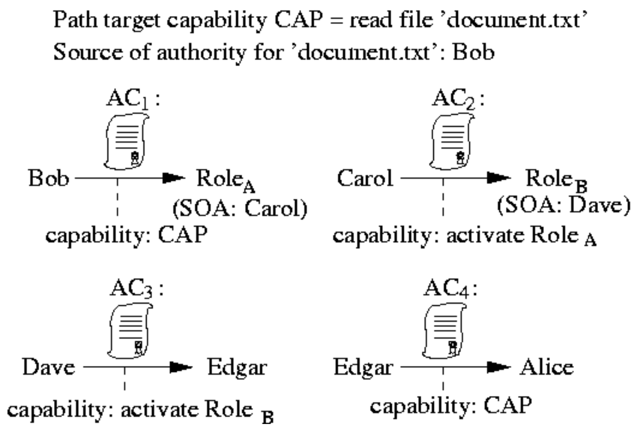
– All certificates in the path must be valid. A certificate is valid if it has a valid signature, it has not expired and it is not revoked.
– Delegation depth limits must be respected all along the path. This applies to both the delegation depth limit of the target and the delegation limits of roles granted in subpaths.
– The path must form an uninterupted chain from the SOA of the target resource to the user requesting access to the resource. Within this chain, there may be subchains, that activate roles to which the resource was granted or that add the resource to some set.

We now explain the complex example of a path shown in figure 3.
Bob is the SOA for the file 'document.txt'. Therefore granting the target capability CAP = 'read document.txt' to the $Role_A$ in AC_1 does not require any other supporting certificates.

Carol, who is the SOA for $Role_A$, can now use CAP. CAP is automatically transferred to $Role_B$ in AC_2. This is because in AC_2 Carol grants activation of $Role_A$ to $Role_B$, thereby making $Role_B$ hierarchically superior to $Role_A$. All permissions of $Role_A$ are inherited by $Role_B$.

Dave, who is the SOA for $Role_B$ grants activation of that role to Edgar in AC_3. This has the following effects: Edgar can now activate $Role_B$. He can subsequently also activate $Role_A$ and may consequently use CAP.

Edgar can now grant CAP to someone else. He does so in AC_4 granting CAP to Alice.



Path target capability CAP = read file 'document.txt'
Source of authority for 'document.txt': Bob
AC1 :
Bob
RoleA
(SOA: Carol)
capability: CAP
AC2 :
Carol
RoleB
(SOA: Dave)
capability: activate Role A
AC3 :
Dave
Edgar
capability: activate Role B
AC4 :
Edgar
Alice
capability: CAP

Fig. 3. A complete, correctly linked path of certificates. See explanations in section 3.5 for details.

The path consisting of AC_1, AC_2, AC_3 and AC_4 therefore grants the target capability to Alice, provided that the delegation depth limits of all ACs are sufficiently high.The Sygn request structure permits to submit multiple paths in a single request. Such a request will be honoured if all paths turn out to be valid (regardless of the fact if it is needed or not). Using this mechanism complex operations requiring multiple permissions can be supported by the Sygn system.

4 Discussion

The following points must be considered when using Sygn:

– Since all users store their own access permissions, there is no central access control service that could be the target of a general denial of service attack. Furthermore since the Sygn access control server runs locally on each storage site, a successful attack on such a server only deactivates access control to the local data. This minimizes the impact of such an attack comparatively to solutions like CAS or VOMS where the access control servers store a relevant amount of permissions. The decentralization of permission storage and access control servers also make Sygn more scalable by spreading the load onto multiple services.
– Access control decisions require no third party. Only the user and the storage site are involved. This speeds up the access decision process and eliminates the danger of corrupted third parties.
– Permission granting does not involve a third party either (except for storing role-related permissions on the review repository). The granting can be done directly between the permission issuer and the AC owner. If such a personal relation does not exist, a second possibility for distribution of Sygn ACs exists: A service like VOMS could be used to distribute Sygn ACs to members of a virtual organization. Since Sygn ACs are protected against tampering and bound to a specific entity, the former concerns against VOMS would no longer apply.

– Sygn supports RBAC. The flexibility of RBAC is perfectly suited for access control in Grid environments especially for applications dealing with complex permission hierarchies.
– The Sygn PDP is completely independent of the Grid middleware, only the Sygn PEP has to be reimplemented for different Grid middlewares, depending on the requests this middleware allows and the used authentication procedures. We have created a Sygn PEP for the μ-Grid middleware [4], that supports basic file access requests like: get-file, delete-file and put-file and uses SSL and X.509 certificates for authentication and communication security.
– Users have to trust the storage sites to enforce their access control directives. Since Sygn is a Grid service and is therefore not integrated in local operating systems, attackers having direct access to a storage site can bypass the Sygn access control mechanism. However users may further protect the confidentiality of their data by encrypting it before storage. We have designed an architecture for managing access to such encrypted data using any Grid access control mechanism (for instance Sygn). This architecture is described in [20].
– The push model of permissions retrieval requires an authorization revokation service. Sygn supports certificate revokation lists (CRL) as used in standard revokation procedures in PKI. Those procedures could therefore easily be adapted to manage this task.
– The security of Sygn depends strongly on the security of the authentication mechanism used in the Grid environment. If certificates and asymmetric keys are used for authentication, an attacker that steals a user's private key can use any ACs that were issued to that user. Therefore measures must be taken to help users that are untrained in computer security to protect their private keys. Using smart-cards to store private keys and perform all private key operations may be a convenient solution.

Sygn answers the requirements as specified in section 1 for a Grid scenario where confidential data is shared among ad-hoc cooperation partners.

Sygn supports role based access control.

It allows fine-grained access control on data and makes individual users owning data the SOA for their data.

It supports delegation of access rights through certificate paths.

Due to the push model and the unique identifiers associated to files, access rights remain consistent even for replicas of these files, provided a user who has access to the files does not voluntarily change the file identifier. Since access rights are not stored with the replicas (only the SOA is), changes of the access rights are effective even if the replica was offline when the change occured.

As the Sygn PDP's are deployed locally on the resource sites, an outage of this specific PDP will only affect the availability of the local resources.

Administrators of hardware resources can use the Sygn resource objects in permissions to regulate access to their resource.

Since permissions can be created and given to users without intervention of a centralized authority, and therefore these permissions are useable at once, ad-hoc permission granting is possible using Sygn.

[4] Available from http://www.i3s.unice.fr/ johan/ugrid/ugrid.html

Finally Sygn provides configurable traceability mechanisms that can even handle non-repudiable tracing of access requests.

Our test implementation was done in the context of the MEDIGRID project[5], that aims at providing a Grid platform for medical image treatment in health-care scenarios. The Sygn PDP is generic and can therefore be used within any Grid architecture. A Sygn PEP for file access control has been implemented for the minimal Grid architecture μ-Grid. It is available (together with μ-Grid) from *http://liris.cnrs.fr/∼lseitz/software*. It provides fine-grained access control to files under μ-Grid and allows automatic retrieval permissions corresponding to user requests from that user's permission repository.

5 Conclusion and future work

We have analyzed the problem of Grid access control and proposed a design which minimizes the impact of successful attacks on components of the access control system. It is intended for Grid environments with high data confidentiality and a low level of trust in the users. An example for such an environment are health-care Grids. Our design responds to both the requirements of the Grid resource sharing scenario and the requirements for a role based access control system. It also provides non-repudiable traceability of all access requests.

A PEP for integration of Sygn in the μGrid middleware exists and we plan to create one for OGSA compliant Grid architectures. We also plan to integrate our encrypted storage mechanisms presented in [20] into the final system.

References

1. Foster, I., Kesselman, C., eds.: The Grid Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc. (1999)
2. Ferraiolo, D., Kuhn, D.R.: Role Based Access Control. In: Proceedings of the 15th NIST-NCSC National Computer Security Conference. (1992) 554–563
3. Pearlman, L., Kesselman, C., Welch, V., Foster, I., Tuecke, S.: The Community Authorization Service: Status and Future. In: Proceedings of the 2003 Conference for Computing in High Energy and Nuclear Physics (CHEP), La Jolla, California (2003)
4. Alfieri, R., Cecchini, R., Ciaschini, V., dell'Agnello, L., Frohner, Á., Gianoli, A., Lörentey, K., Spataro, F.: VOMS, an Authorization System for Virtual Organizations. In: Proceedings of the 1st European Across Grids Conference, Santiago de Compostela, Spain (2003)
5. Thompson, M., Johnston, W., Mudumbai, S., Hoo, G., Jackson, K., Essirari, A.: Certificate-based Access Control for Widely Distributed Resources. In: Proceedings of the 8th USENIX Security Symposium, Washinton, D.C., USA (1999)
6. Chadwick, D., Otenko, A.: The PERMIS X.509 Role Based Privilege Management Infrastructure. In: Proceedings of the 7th ACM Symposium on Access Control Models and Technologies, Monterey, CA, USA (2002) 135–140
7. Farrell, S., Housley, R.: An Internet Attribute Certificate Profile for Authorization. Request For Comments (RFC) 3281, Internet Egnineering Task Force (IETF) (2002) http://www.ietf.org/rfc/rfc3281.txt (Webpage visited on 12/04/05).

[5] http://creatis-www.insa-lyon.fr/MEDIGRID

8. Erdos, M., Cantor, S.: Shibboleth-architecture draft v05. Technical report, Internet2 (2002) http://middleware.internet2.edu/shibboleth (Webpage visited on 12/04/05).
9. Vollbrecht, J., Calhoun, P., Farrell, S., Gommans, L., Gross, G., de Bruijn, B., de Laat, C., Holdrege, M., Spence, D.: AAA Authorization Framework. Request For Comments (RFC) 2904, Internet Engineering Task Force (IETF) (2000) http://www.ietf.org/rfc/rfc2904.txt (Webpage visited on 12/04/05).
10. Lepro, R.: Cardea: Dynamic Access Control in Distributed Systems. Technical Report NAS-03-020, NASA Advanced Supercomputing (NAS) Division (2003)
11. Godik, S., Eds., T.M.: eXtensible Access Control Markup Language (XACML). Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003) http://www.oasis-open.org/ (Webpage visited on 12/04/05).
12. Maler, E., Mishra, P., Eds., R.P.: The OASIS Security Assertion Markup Language (SAML) v1.1. Standard, Organization for the Advancement of Structured Information Standards (OASIS) (2003) http://www.oasis-open.org (Webpage visited on 12/04/05).
13. Lorch, M., Kafura, D.: Supporting Secure Ad-hoc User Collaboration in Grid Environments. In: Proceedings of the 3rd International Workshop on Grid Computing, Baltimore, MD, USA (2002)
14. Lorch, M., Adams, D., Kafura, D., Koneni, M., Rathi, A., Shah, S.: The PRIMA System for Privilege Management, Authorization and Enforcement. In: Proceedings of the 4th International Workshop on Grid Computing, Phoenix, AR, USA (2003)
15. McNab, A., Kaushal, S.: Gridsite: Grid access control language. http://www.gridsite.org/1.0.x/gacl.html (2003) (Webpage visited on 12/04/05).
16. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI Certificate Theory. Request For Comments (RFC) 2693, Internet Engineering Task Force (IETF) (1999) http://www.ietf.org/rfc/rfc2693.txt (Webpage visited on 12/04/05).
17. Ferraiolo, D., Sandhu, R., Gavrilla, S., Kuhn, D.R., Chandramouli, R.: A Proposed Standard for Role Based Access Control. ACM Transactions on Information and System Security **4** (2001)
18. PKIX Working Group: Public Key Infrastructure (X.509). Technical report, Internet Engineering Task Force (IETF) (2002) http://www.ietf.org/html.charters/pkix-charter.html.
19. Seitz, L., Pierson, J., Brunie, L.: Key management for encrypted data storage in distributed systems. In: Proceedings of the second Security In Storage Workshop (SISW), Washington, D.C., USA, IEEE Computer Society (2003) 20–30