A Grid Certificate Authority for Community and Ad-hoc Grids*

Gregor von Laszewski
[1]Argonne National Laboratory
9700 S. Cass Ave.
Argonne, IL 60439, U.S.A.
[2]University of Chicago, Chicago, IL
gregor@mcs.anl.gov

Mikhail Sosonkin,
[3] Polytechnic University
6 Metrotech Center
Brooklyn, NY 11201, U.S.A.
mike@isis.poly.edu

Abstract

One of the most important issues in Grid computing is to provide a secure environment that allows administrators to contribute their resources and users to utilize them. Currently diverse methods are required to obtain certificates for the different Grids. In this paper we showcase a prototype of a tool that simplifies the tasks associated with maintaining a Grid certificate authority and simplifies the application process for the user to interact with multiple certificate authorities.

1. Introduction

The Grid approach [1] provides a vision for the utilization of multiple physically dispersed computing resources. Grid services unite the resources and provide the user with access to seamlessly integrated resources. In order to utilize these resources, production Grids employ a trust model. The following two trust models are most common. In a *community production Grid*, members provide resources and allow users of the community to use the resources once the users are trusted and obtain authorization to use the resources. In a *volunteer production Grid*, users donate unused computational cycles to achieve, (most often), a nonprofit scientific task [2]. The membership is based on an implicit trust model established through an inverse security assurance. While in a traditional community Grid the users run their applications on trusted resources, in a volunteer Grid the resource contributors execute trusted applications. In the paper we focus on community production Grids that restrict access to its resources through an account application process.

Establishing trust to such a Grid requires special efforts. Grid middleware provides the resources with secure communication methods to prevent exploitation of the communication between clients and services. In order to establish such secure communication, various methods exist. For example, *symmetric key cryptography* is used to secure communication from source to destination. This type of cryptography uses the same key for both encryption and decryption of data. The mechanism is secure only when both parties have exchanged the keys using another secure mechanism. A convenient mechanism used for secret key exchange is *asymmetric cryptography*. It uses one key for encryption and another for decryption. To complete this process, one needs a way of linking a key to an entity that it belongs to. The process involves a third party, the Certificate Authority (CA). The authority can issue a certificate that only the authority can modify. The certificate uniquely identifies the CA. Systems and users that trust this CA can obtain certificates that are signed by this CA. The signing process is usually well documented and is cause for the trust based into the CA. In order to increase the trust, in many systems a hierarchical structure of CAs is used. In Figure 1 we depict such a system, where the most trusted authority is at the top and the least trusted authorities are at the bottom. The root authority is trusted by a variety of other less trusted authorities. The authorities at the bottom are trusted only to issue certificates to clients but are not trusted by other authorities. A client may need multiple certificates in order to interact with different entities that are not part of the same trust domain; in other words authentication is governed by different CAs.

While reviewing a variety of community Grids we observed that few tools are readily available for the process of maintaining a Grid CA and simplifying the process of

*7th International Workshop on Javatm for Parallel and Distributed Computing, held in conjunction with the 19th International Parallel and Distributed Processing Symposium (IPDPS 2005) April 4-8, 2005, Denver, CO, USA. This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38 and by the National Science Foundation under Grant No. 0353989.

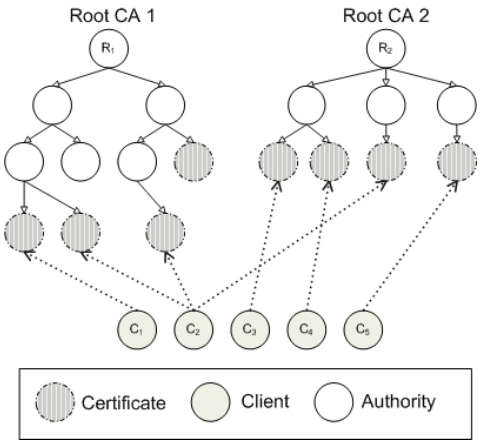

Figure 1. Illustration of the hierarchal nature of certificate authorities and certificates.

obtaining a user certificate. We believe that this situation is a cause of concern since it prevents the widespread deployment of Grids. Users are faced with complex application processes that make the use of a community production Grid to cumbersome to use for users with limited information technology knowledge. Administrators are faced with the burden of issuing many certificates taking a considerable time.

Hence, with our proposed solution we address specifically the following issues:

1. Make it easy to set up a CA on behalf of a community.
2. Make it easy to maintain certificate requests from the community.
3. Make it easy for the community members to obtain certificates and place them on a client machine.
4. Make it possible to obtain certificates from multiple CAs if desired.

2. Requirements

To design our CA toolkit appropriately, we take a closer look at a number of requirements that have immediate benefit to our target audience.

Different user identities. One important factor in setting up a CA is the verification of the users identity as part of a community. The application process typically involves the gathering of data from the user. In Table 1 we have listed a number of such fields as they are required by CAs that spawn a large number of Grid users. These CAs include DOE [3], NCSA [4], NPACI [5], Access Grid (AG) [6], the Java CoG Kit CA applets [7], and the Simple CA [8]. We observe that certain fields overlap and some are not required to submit a request.[1]

Additionally we observe systems that are tightly integrated with an existing account application process such as the NCSA or NPACI user ID and accounts. Although this table is not exhaustive, it gives a general idea of what the administrators of CA's want from the user for authentication and authorization.

From this table and from the analysis of existing Grid middleware toolkits, it is clear that we do not yet have convenient middleware tools to help setting up certificate authorities with slightly different identity requirements. The current generation of tools, including those provided by community projects such as ESG [9] and Fusion Grid [10], provide only a relatively fixed solution with little flexibility in setting up easily different identity requirements. That such individualization is necessary is obvious from efforts documented as part of the the European Policy Management Authority for Grid Authentication (Eu Grid PMA) [11] and the list of authorities at [12]. Once a CA has been designed, it can been reviewed based on the policies that are going to be gathered in [11].

Table 1. Fields needed to make a request to various CAs

Field Name	DOE	NCSA	NPACI	Access Grid	Java CoG Kit	Simple CA
Full Name	×			×	×	
Email Address	×			×	×	
Contact Email	×					
Contact Phone	×					
Affiliation	×					
Name of Sponsor	×					
Sponsor's Email	×					
Additional Info	×					
Key Length	×					
NCSA User ID		×				
NCSA Kerberos Password		×				
NPACI Workstation Account			×			
Certificate Type				×		
Domain				×		
Organization					×	
Organization Unit					×	
Country					×	
Common Name						×

[1] We have not listed fields that are not required for the establishment of the identity.

Multiplicity of CAs. Since different Grid communities have different authoritative requirements, as discussed in [13], users are frequently part of multiple communities. (In fact, the first author of this paper is in three different communities requiring different certificates for each of the Grids. This situation poses a significant problem to users that are not as technology aware.

Simple use. One of the important features of the tool must be its ease of use for users and administrators. A graphical user interface (GUI) is needed to simplify the certificate application by users and administrators.

Simple to deploy. The tool must be easy to deploy for the same reasons given in the preceding requirements.

Simple to enhance. The tool must be simple to enhance because adaptations for various community Grids will be necessary.

3. Design

The design of our tool to develop a Grid CA is based on a client-server model that uses one server and multiple clients (Figure 2). Our design should not be confused with the applets that are already distributed with the Java CoG kit and can interface with a myProxy maintained CA [14] and are referred to in Table 1.

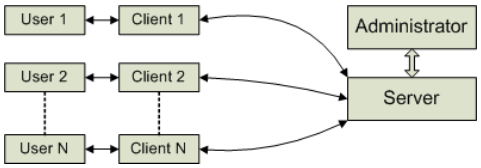

Figure 2. Client-server model of the system

Figure 3 shows the high-level design of the system and depicts interactions between the user and administrator and the different components that are part of the system. A user can request a certificate from an administrator by using the GUI that is part of our client. The client will contact on behalf of the user to the certificate server and retrieve a number of fields that need to be completed and are defined by the CA administrator. Once the user has completed the information and approves its submission, the client GUI forwards the information to the client certificate manager. The client certificate manager saves the information in a secure local database and generates the request. The client certificate manager then uses a connection to the CA server to send the request to the administrator. The CA server accepts the request and saves it in the server database. When the administrator returns to his workstation, the request will be read from the database by the specialized administrator interface and displayed. At this point the administrator can either sign the request and generate a certificate for

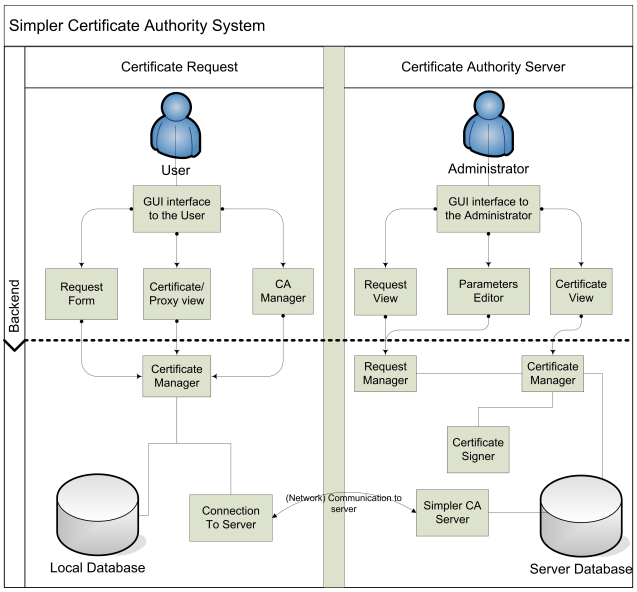

Figure 3. High-level design of the system

the user requesting it or deny it. If the administrator signs the request, the certificate is saved in the server's database. The administrator might notify the user that the certificate is ready through some other agreed on protocol (such as e-mail), or the user simply may periodically try to retrieve the certificate until it's available. When the users client receives the certificate it is stored in the local database. In order for the system to work, the client needs to know the address of the certificate server. This can be integrated in our distribution or a custom distribution can be provided making the contact implicit. This adaptation is easy because our code is based on object-oriented technologies that allow adaptation through Java interfaces and their implementation.

With the help of this tool the steps can be implemented as part of a client portal that simplifies the certificate request. Variations to these steps are possible and depend on the communitys requirement.

1. Go to the Grid community portal you are a member of.
2. Go to the certificate request page.
3. Click on request a certificate.
4. The portal will now install or invoke the client request program and connect to the appropriate CA.
5. The client retrieves fields necessary to conduct the identification step.
6. Fill out the request form.
7. Press the submit button to send the request to the CA.
8. A mail is sent to the user that the certificate is ready for pickup.
9. Press the retrieve button to obtain the certificate.

In comparison to the procedure that is necessary for retrieving a DOE certificate [3], for example, our process appears to the user significantly simpler. Less technical expertise is necessary to complete the process, and the user has to focus only on filling out the form, submitting it, and retrieving the certificate at a later time through a simple button click.

To set up a certificate authority suitable for the community, the CA designer can configure the fields necessary to establish identity. Once the CA is installed, it accepts incoming requests from clients and stores the incoming request till they are managed by the administrator. Once the administrator connects securely with the GUI to the CA, the approval process can be easily implemented as follows for each certificate request.

Steps for the CA to generate the certificate include:

1. Verify the identity of the person conducting the request based on the information provided by the requestor.
2. If the request is denied, end this request
3. If the request is approved, press a button to continue.

Internally the certificate is generated and placed in a special location for pickup and an optional mail is created to inform the user that the certificate is ready for pickup.

4. Implementation

We have implemented the system as specified in our initial design.

Certificate authority management. The certificate authority is administered through a simple GUI as shown in Figure 4 and 5. The administrator can easily specify the fields used for the user identification (Figure 4). The order of the fields can be changed, new fields can be added, and fields can be marked as required to be filled out by the client before a request can be submitted to the CA. The fields and their contents can be included as part of a certificate. Outstanding certificate requests can be viewed and approved through a convenient approval panel (Figure 5). The data on the server side is organized in a hierarchical file system. It contains directories for requests that are newly received requests that have been read, but not yet completed requests that have been signed and requests that are denied. Each request is assigned a unique ID. The server is designed to be multithreaded in order to handle multiple incoming requests from clients.

User certificate management. The convenient client GUI lets users view CAs, certificates, and proxies in the local database, using a tree structure (Figure 6) representing an entry for each CA. Below each CA is the list of all the certificates and requests. A certificate is augmented by a check box that represents the validity of he certificate. Below each certificate are listed the proxies associated with the

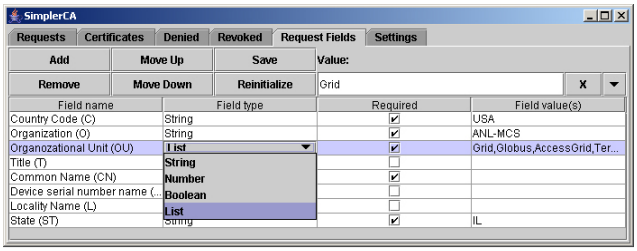

Figure 4. Administrator request fields editor

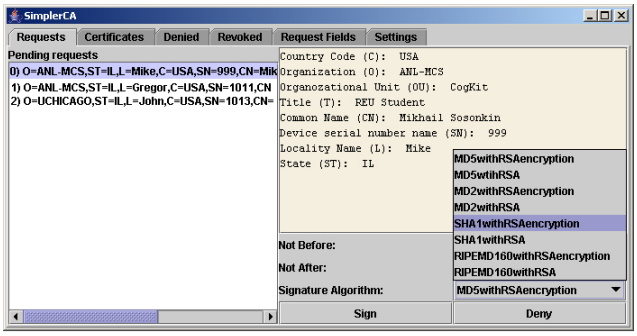

Figure 5. Administrator request list viewer

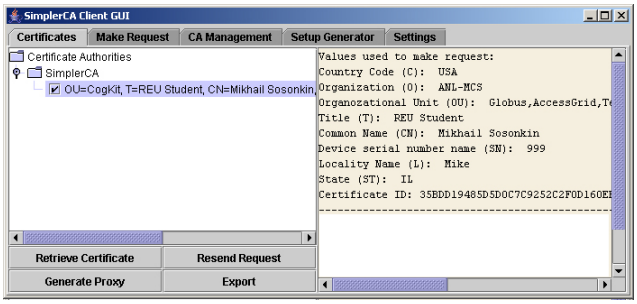

Figure 6. User certificate and proxy viewer

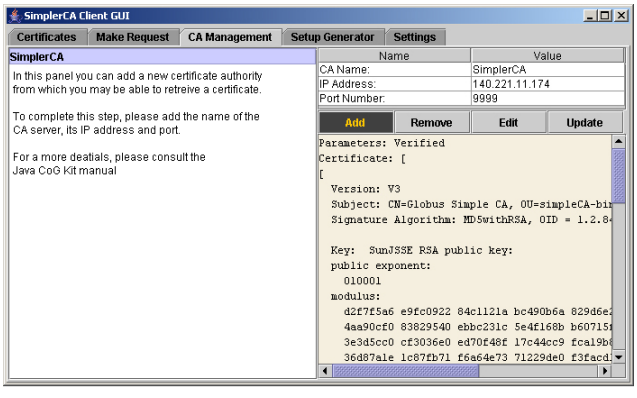

Figure 7. User certificate authority manager

certificate and their validity is identified through a check in the associated checkbox. New certificate authorities can be added simply by invoking the CA management panel (Figure 7). After the CA server has been added, the fields are retrieved from the remote location. Once the CA is included, and the required fields have been filled out by the user, the certificate form the CA can be requested. The information stored by a client is organized hierarchically on the operating system's file and directory system in an especially assignable directory. A number of configuration files determine the CA server, the fields that need to be filled out, the requests that have been issued, and the signature of the configuration and can be identified by an MD5 hash.

Data Formats. To support the communication between client and server, we use a number of data formats. A request contains a specially marked PKCS10 certificate request and a CA envelope, both encoded in Base64. The envelope is created because that data needs to be sent to the server confidentially. The private key is stored in a Based64 encoded PKCS8 format. However, we have enhanced our framework to support multiple formats, such as PKCS1, which can be used if the CA designer requires. Signatures of the server and client parameter file are stored in PKCS7 format in order to protect the data from unauthorized tampering. We have designed special Java objects that abstract these data formats and provide a convenient interface of handling the data. For a short description about the intent of these formats, we refer the reader to [15].

Network Communication. Figure 8 shows the messages that are passed between the client and the server when the client adds a new CA retrieves the information about this CA from the server. A client request to retrieve a certificate is depicted in Figure 9 according to our design discussions from Section 3.

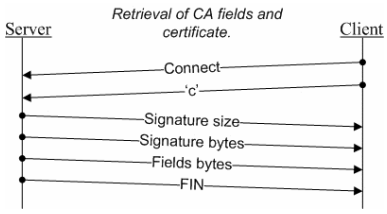

Figure 8. Client-server communication to receive the CA certificate and fields from the CA.

Information Security. Since the information gathered from the identity form may contain sensitive information, it is important to secure the location in which that information is stored and to secure the communication path between the client and server. The latter is achieved by using encryption

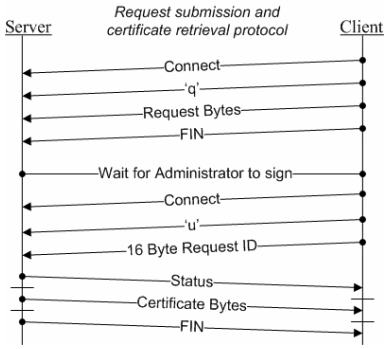

Figure 9. Client-server communication to receive a certificate.

on the socket and using a PKCS5 padding scheme. To avoid blocks being created with the same value during encryption, we use a cipher block chain (CBC) mode [16] to encrypt a message.

At present we have implemented the strategy to obtain the servers certificate through a secure download process. To limit the risks of a man-in-the-middle attack, however, we intend our final implementation to distribute the public key as part of our signed software distribution. This way we can assure confidence in establishing a direct connection between the client and the server.

Program Design. The graphical interface and the actual implementation are cleanly separated; the design is such that there is oneway communication instantiation between the GUI and the backend. That is, the backend has no knowledge of how its features are presented to the user. This design allows for an easy way of replacing the GUI or automating tasks. All of the functions are accessed by invoking methods that are defined by an interface for a particular component. The design style is applied to both the client and the server.

For example, the connection from the client to the server is abstracted through a convenient interface as listed in Listing 1.[2] This interface is reused as part of the GUI component. Hence, if improvements to the connection handling or the GUI are necessary, the impact on the overall code by modifying these components is minimal.

The client and server use an event model to send communications to the client's GUI. The listeners are used to notify components that need to be informed if requests have been issued. The client and server are checkpointable to allow graceful shutdown and startup. The user could, for example, start filling out the identify fields and return at a later

[2]For space saving reasons, we have not included Exceptions in our code examples. Hence they represent pseudo codes and not the real interfaces. These can be obtained from the www.cogit.org Web page.

Listing 1. Pseudo Interface for a connection to CA.

```
package org.globus.cog.security.management.client;
public interface CAConnection {
  public void sendRequest(Request req)
  public Certificate getCertificate(Request req)
  public byte[] getParameters()
  public byte[] getSignature()
  public void write(OutputStream out)
  public void read(InputStream in)
}
```

time to submit or retrieve the request. For the system administrator it provides the added benefit that he can interrupt the approval process at any time and revisit it later. Hence our system more closely adapts to the process users naturally interact with the Grid instead of forcing the user to conduct an uninterruptible process. We believe that through our system the user experience with a CA is much enhanced.

5. Conclusion

We introduce a new system that provides a Grid certificate authority. The system is targeted to design, deploy, and use a manageable system to handle certificates from the point of view of the user and Grid administrator. As our system is expandable, enhancements are easily integratable, such as the deployment of keys as part of grid map files. The main design goals have been supported through appropriate use of object-oriented programming as well as a preliminary graphical user interface that hides much of the complexities. Because our system is easy to use and install, it supports the sporadic and ad-hoc nature of many research collaborations. We have started to improve our GUI to allow the deployment through Java Webstart and to provide a customizable GUI interface for the client that focuses on just one CA to support a much more simplistic user clientele. We emphasize, however, that many of our advanced users have the need to access multiple CAs, a requirement that is already supported by our software.

It would be conceivable to enhance the system in such a way that the user must never need to think about certificate authorities or certificates. This could be achieved by integrating a simple request procedure as part of the setup component of the Java CoG Kit that has for years provided a simple way to interact with Grids. We as a project would be willing to work with other Grid projects together to develop appropriate deployment modules customized for the particular Grid community.

The Java CoG Kit has an abstraction layer to include besides GSI multiple other security frameworks such as SSH. Our CA does not have at this time the ability to work with different security providers. The focus at this time is on GSI.

References

[1] G. von Laszewski and P. Wagstrom, *Tools and Environments for Parallel and Distributed Computing*, ser. Series on Parallel and Distributed Computing. Wiley, 2004, ch. Gestalt of the Grid, pp. 149–187. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf

[2] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky, "SETI@home-massively distributed computing for SETI," *Computing in Science & Engineering*, vol. 3, no. 1, pp. 78–83, January–February 2001.

[3] "DOEGrids Certificate Service," Webpage, June 2004. [Online]. Available: https://www.doegrids.org:443/

[4] "NCSA CA Client Software," Webpage, June 2004. [Online]. Available: http://grid.ncsa.uiuc.edu/ca/client/

[5] "NPACI HotPage Grid Computing Portal," Webpage, June 2004. [Online]. Available: https://hotpage.npaci.edu/

[6] Webpage, June 2004. [Online]. Available: http://www.accessgrid.org/

[7] Jean-Claude Cote, "NRC Certificate Applets," Webpage, also distributed as part of the Java CoG Kit., June 2004. [Online]. Available: http://www.cogkit.org

[8] "Globus Toolkit Simple CA," Webpage, June 2004. [Online]. Available: http://www.globus.org/security/simple-ca.html

[9] "Earth System Grid CA," unpublished. [Online]. Available: https://www.earthsystemgrid.org

[10] "Fusion Grid CA," unpublished. [Online]. Available: http://www.fusiongrid.org/

[11] "European Grid Policy Management." [Online]. Available: http://www.eugridpma.org/

[12] "Certificate Authorities," Webpage, August 2004. [Online]. Available: http://marianne.in2p3.fr/datagrid/ca/ca-table-ca.html

[13] G. von Laszewski and K. Amin, *Grid Middleware*. Wiley, 2004, ch. Chapter 5 in Middleware for Commnications, pp. 109–130. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf

[14] G. von Laszewski, B. Alunkal, K. Amin, J. Gawor, M. Hategan, and S. Nijsure, "The Java CoG Kit User Manual," Argonne National Laboratory, MCS Technical Memorandum ANL/MCS-TM-259, Mar. 14 2003. [Online]. Available: http://www.mcs.anl.gov/~gregor/papers/manual-cog2.pdf

[15] RSA Laboratories, “PKCS Overview,” Webpage. [Online]. Available: http://www.rsasecurity.com/rsalabs/node.asp?id=2124

[16] W. Mao, *Modern Cryptography Theory and Practice*. Prentice Hall, 2004.