Explicit Trust Delegation: Security for Dynamic Grids

● David F. Snelling ● Sven van den Berghe ● Vivian Qian Li

(Manuscript received June 24, 2004)

This paper addresses the issue of how to build dynamic Grids without using the non-standard proxy extensions that cause concern within the security community. The approach allows commonly available security tools and libraries to invoke requests (on the client side) and respond (on the server side) using only well-established security protocols. This description is made in the context of the UNICORE Grid infrastructure. UNICORE is known to have a strong, respected security model, but at the cost of not supporting some dynamic Grid capabilities. The discussion shows how UNICORE could be enhanced using Explicit Trust Delegation to provide dynamic capabilities, hitherto only possible in Grids supporting a proxy-based security model. We show how this approach provides a smooth migration path to proposed work on Virtual Organizations.

1. Introduction

1.1 Need for delegation

Grid computing is built on the possibility of users combining a number of different resources into a single job to perform some action on their behalf. The resources used for the job may come under the control of different organizations and could be spread over a number of different locations, and their use will be spread over time. The sites that run the parts of the job will want to be able to know that the work was authorised by a particular user so the appropriate charges can be made and the security of the sites assured. However, the user is disconnected from the execution of the job (in space and time) and so cannot be asked directly for authorization of a particular action. Complex job flows in a Grid application will also mean that two servers might need to communicate directly with each other (without the user being present).

This is in contrast to the World Wide Web, where interactions are always between an individual and a Web server and the individual is always "present" to authorise an action if necessary. Traditional security mechanisms for authentication and authorization such as the Public Key Infrastructure (PKI) -based authentication used with https work well when the client is connected to the service.

When the client (end-user) becomes disconnected from the server, a more complex situation arises. The server must operate without direct authorization since it can no longer contact the user. There can be many reasons for this logical disconnection; for example, the end-user may physically disconnect from the server, the task may operate in a batch-like environment, or the server may create tasks remotely from the end-user.

A common solution to the problem of disconnection is delegation, whereby end-users pass on (grant) rights to a server (or process) to perform

tasks on their behalf.

The traditional WWW interaction uses "direct authorization", whereby the end-users are connected to a process when they request it to perform an action and so can authorize the action at the time it is to be performed. "Delegation" is the process whereby one entity on the Grid (usually an end-user) grants rights to another entity on the Grid (usually a process) to perform actions on the entity's behalf. "Static delegation" occurs when the actions to be authorized are known in advance of the delegation, so the delegation can be limited to just the required actions and no more. "Dynamic delegation" refers to the delegation of rights to perform actions that are not known at the time a task is started. Dynamic delegation can support the runtime selection of the site or machine on which to run a task and is done, for example, when performing global load balancing, brokering, or conditional computation within a workflow graph.

Both types of delegation imply that the rights grantor cannot provide direct authorization because the grantor is disconnected from the grantee when the actions are executed.

There is always a compromise between flexibility and security in any distributed computing context, but this is particularly true in Grid computing. The flexibility of the system is reduced whenever, in the interest of increased security, a constraint is placed on the tasks that can be performed by a Grid entity. Security is all about getting that balance right, and Grid computing raises more complex issues in this area than traditional uses of the Internet.

The most flexible form of delegation is impersonation, whereby the rights grantor allows the receiver to assume the identity of the grantor and perform any action on the grantor's behalf. However, this is also the least secure, since the delegated rights could be easily abused. Therefore, delegation should be limited to the work that is required.

1.2 Application of delegation

An example of the distinction between direct and delegated authorization is the process of executing jobs through a UNIX shell. When logged into a UNIX shell, the end-users pass authority to the shell to run processes on their behalf. While an end-user is connected to the shell, the shell has direct authorization. When an end-user becomes disconnected (e.g., because the end-user puts a process into the background and logs out of the shell), the authorization becomes delegated. Actions performed by the backgrounded process are acting under delegate authorization.[note 1)] More interesting examples emerge in the context of Grids.

Consider an application portal based on a PKI for authentication. The portal provides a secure interface to the end-user over the Internet and interacts with the end-user to create a task description of some sort (in UNICORE this would be an Abstract Job Object [AJO]). The portal then submits the task description to a server for execution. The server will require some authority to execute the described task, and this must come from the end-user and not the Portal. The creation and execution of the task description, if performed while the end-user is connected to the portal, could be performed under direct authorization from the end-user. This is because mechanisms exist that allow end-users to establish that they trust the portal to create (and even run) tasks their behalf. If the portal must pass a task (or sub-tasks) on to other Grid entities, some form of delegation must be applied so the entity running the task can ascertain that the end-user actually requested that the task be performed.

One way of delegating the authority is for end-users to create digital signatures of task descriptions using their private keys (see below).

note 1) Because there is a security risk associated with disconnected processes, many system's administrators prevent end-users from running background processes when they are not logged in.

There are two methods by which such a signature can be applied to a task description: 1) the task description is returned to the end-user for inspection and signing or 2) the portal software signs it on behalf of the end-user.

If the tasks to be performed are known while the end-user is connected, the task description can be authorized by static delegation using method 1). In general, this is an unsatisfactory solution because the purpose of a portal is to hide the details of the task description from the end-user.

Method 2) is more satisfactory because it hides the details of the task description from the end-user and also allows for changes to the task description after the end-user has disconnected.

In UNICORE Version 4, only static delegation is possible. The aim of Explicit Trust Delegation is to extend UNICORE's functionality to include dynamic delegation.

1.3 Trust issue

Various forms of delegation exist in many (non-Grid) systems today. For example, a batch sub-system reads a job script and creates processes on the system on behalf of the end-user who submitted the job. Incoming mail is written to an inbox owned by the end-user. In these cases, the systems administrator authorizes the server processes to perform these actions, usually through processes that run continuously with more privileges than a normal user. This basic strategy has proven effective within a single administrative domain and is commonly enhanced with various extensions, for example, complex collections of roles, access control lists, and time-expiring access tickets. In the end, all these mechanisms require that the end-user trust the system's administration.

In a Grid, the trust relationships required to execute a task become much more complicated. For example, a server at one site may want to write to a file at another site on behalf of an end-user. However, it is unlikely that the destination site will trust the source site to write the file directly on behalf of the end-user. Therefore, the source site passes a message saying, "Please write this file for me on behalf of this end-user." If the destination site is disconnected from the end-user, it has no way to check that the end-user actually authorized the file to be written.[note 2)] The spectrum of possible trust relationships is quite large. At one end of this spectrum, the destination site could trust the source site to request that a file be written on behalf of any end-user at any time. At the other end, the destination site could distrust the source site completely and only write the file when a connection to the end-user can be established to authorize the action directly.

The UNICORE security model supports this type of function using static delegation: the request to transfer a file from the source site to the destination site is signed by the end-user when the job is created. This approach requires a minimal trust relationship between the sites (since the end-user has explicitly authorised the file transfer request). However, there is no support for dynamic Grid functions (e.g., file transfer from a source whose location is only discovered during execution) since the entire job tree must be signed by the end-user at the client when the job is created.

Another approach to gaining authorization is to reconnect the end-user to the destination site through the proposed development in Virtual Organizations (VOs).[1)] A VO is a trusted third party that mediates between end-users and agents (e.g., servers) of the Grid to establish trust relationships. VOs would provide dynamic support for joining and leaving. Once an end-user becomes a member, the end-user and agents can obtain trust assertions from the VO server.

In the remainder of this paper, we describe

note 2) There is of course the reciprocal trust relationship that the source site trusts the destination site to actually write the file. We omit this side of the relationship as it requires a richer set of security mechanisms than is in common use in the Grid today, e.g., non-refutation.

how UNICORE will be enhanced using Explicit Trust Delegation to provide these dynamic Grid capabilities and how this approach provides a smooth migration path to this proposed work on VOs. In the next section, we describe some of the background in security, including general issues and specifics of the UNICORE model and the Grid Security Infrastructure (GSI) from Globus[2] and how each balances the requirements of dynamism with the needs of security. We then outline the new Explicit Trust Delegation framework for UNICORE and how it reconciles the conflict between dynamism and security. We conclude with related work and future plans for security services within the UNICORE Grid.

2. Background

2.1 Introduction to PKI and SSL

The ability to perform work on remote sites requires secure authentication of users so that the remote site can know unambiguously who is trying to perform the work. Grids usually establish identity using a PKI based on X.509 certificates. The entities in a PKI (i.e., end-users and servers) are identified by a globally unique name called the Distinguished Name (DN). Entities prove their identity by possessing a set of credentials consisting of an X.509 certificate and a matching private key. The X.509 certificate binds the entity's DN to the private key. Entities authenticate by presenting their certificate and then proving that they know the corresponding private key. Certificates are public, and so it is knowledge of the private key that establishes an entity's claim to an identity. This means that if a private key is known to another entity, that entity can impersonate the private key's owner at will and without restraint. Therefore, private keys must be kept secret and are normally stored in an encrypted file whose decryption pass phrase is known only to the end-user.

Certificate Authorities (CAs) issue X.509 certificates. CAs are trusted by resource owners to reliably establish the identity of end-users and will have procedures in place to do this. The existence of a CA and its procedures to establish an identity means that obtaining an X.509 certificate is an action that should be performed infrequently. X.509 certificates typically have lifetimes of the order of years.

Communication between entities in a Grid is performed using the SSL protocol to secure the communications. SSL communications use the credentials provided by a PKI to build a secure communication channel. The secure channel provides authentication because each end of the channel knows the other end's identity. SSL can also provide integrity (a guarantee that the message has not been changed by anyone else during transmission) and privacy (a guarantee that no one else can read the message during transmission). SSL is a standard protocol available in almost all Internet-enabled applications.

2.2 Static-Grid delegation by SSH

Part of the aim of delegation is to execute work on a remote system using secure authentication. This can be accomplished most simply by using Secure SHell (SSH). SSH can be configured to use RSA[note 3] authentication, which, like the PKI described above, is based on public key cryptography. However, unlike the PKI, it does not have a globally verifiable identity (i.e., it does not have a DN bound to a public certificate). SSH end-users must identify themselves to each site at least once by using another method (e.g., user name and password). Following this initial login, SSH can be configured to allow subsequent logins to rely on password-less RSA authentication. It is thus possible for an end-user to set up a private Grid of sites that can be coordinated to execute work. However, such a Grid will be inherently static because it is limited to the sites that the end-user

note 3) RSA is a public key cipher, which can be used both for encrypting messages and making digital signatures. The letters stand for the names of the inventors: Rivest, Shamir, and Adleman.

has explicitly set up and as such will be difficult to manage.

In a chain of sites thus organized, the SSH implementation at one site has been delegated authority (by the end-user at the initial login) to perform actions at another site on the end-user's behalf. Should the SSH implementation or the end-user's account become compromised at the first site, the end-user could be impersonated at the second site. For this reason, sites must trust each other's internal security policies and administrators. Grids based on such mechanisms require that all sites agree to a minimum level of security and thus establish a notion of "transitive trust" among themselves.

2.3 UNICORE V4 model

Actions to be performed on a UNICORE Grid are described in a Java object called an AJO. The classes making up an AJO provide support for many types of computational tasks, job control and management, file transfer and management, and also complex functions such as loops and conditional computation. Multi-site jobs are created by including sub-AJOs within a parent AJO. An end user sends an AJO to a UNICORE server for execution. Sub-AJOs in multi-site jobs are consigned from server to server. The process of sending an AJO from end-user to server or from server to server is known as "consigning".

AJOs are consigned using an SSL-based protocol called the UNICORE Protocol Layer (UPL). The UPL provides secure, authenticated communication between the UNICORE clients and servers, including high-speed file transfer. The UPL's secure connection is based on the same X.509 certificates used for identifying end-users and servers in a UNICORE Grid.

The consign part of the UPL requires two certificates to be transferred with every AJO: the "endorser" certificate and the "consignor" certificate. These certificates represent different roles in the delegation process. The endorser is the end-user to whom the AJO belongs (i.e., the person who created the AJO and under whose authority the AJO will be executed). Ownership of the AJO is established by transferring to the server a digital signature of the AJO along with the AJO itself. This digital signature is created using the private key bound to the endorser certificate. A valid digital signature of an AJO is taken to be authorization by the end-user for the actions described in the AJO to be performed by the server. The consignor is the agent that transfers the AJO to the server and is either the end-user or another UNICORE server (sub-AJOs). The consignor certificate is extracted from the UPL connection used to transfer the AJO.

When an end-user consigns an AJO, the end-user's certificate is used for both the Consignor and Endorser roles (**Figure 1**). When a sub-AJO is consigned between two servers, the end-user's certificate has already been used to endorse the sub-AJO and the server's certificate is used to establish the UPL connection and thus consign the sub-AJO (**Figure 2**). An end-user is allowed to endorse and consign AJOs, whereas a server can only consign AJOs. This means that server processes cannot create jobs. A server can only consign jobs that have been endorsed by end-users. This allows servers to consign sub-jobs, but these sub-jobs will be executed with the delegated identity of the Endorser.[note 4)] End-users may only consign jobs that they have endorsed.

These rules mean that sites running UNICORE do not need to trust other UNICORE sites in order to receive sub-AJOs from them. All AJOs they receive must have been created by end-users and cannot be modified by intermediary servers.

Once an AJO is received, the server checks that the Endorser has properly signed the AJO. Following this, the server performs local site authorization, ensuring that the Endorser is allowed to execute jobs on the site and is mapped to a local account.

The current UNICORE delegation model is static. The end-user describes the steps that a

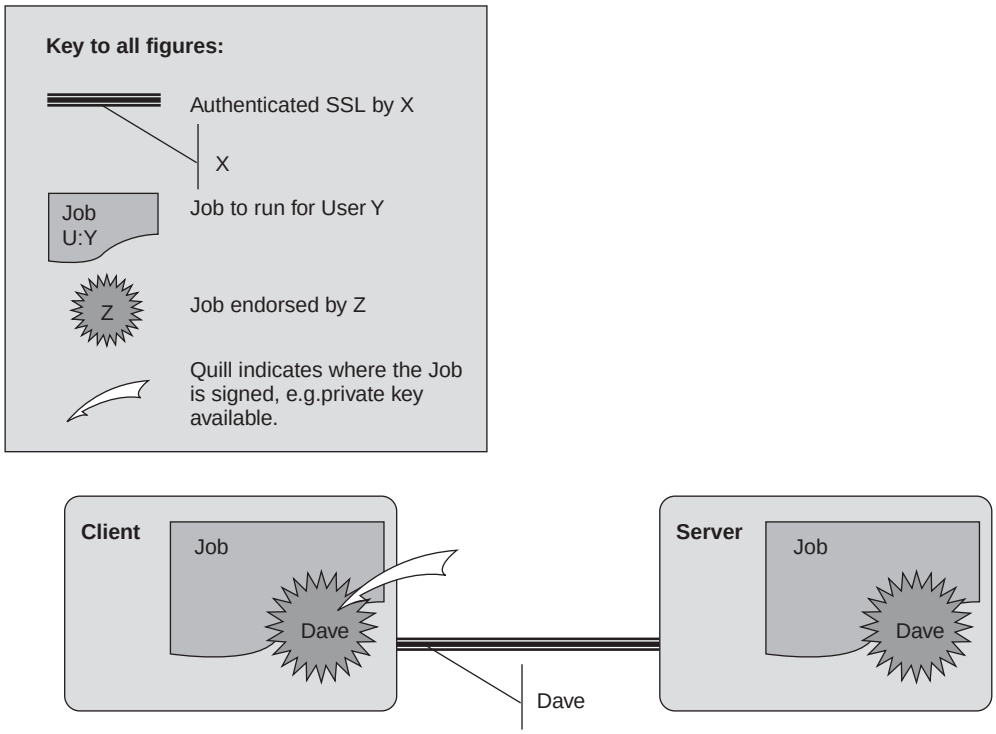

Figure 1
UNICORE V4 single-site job.

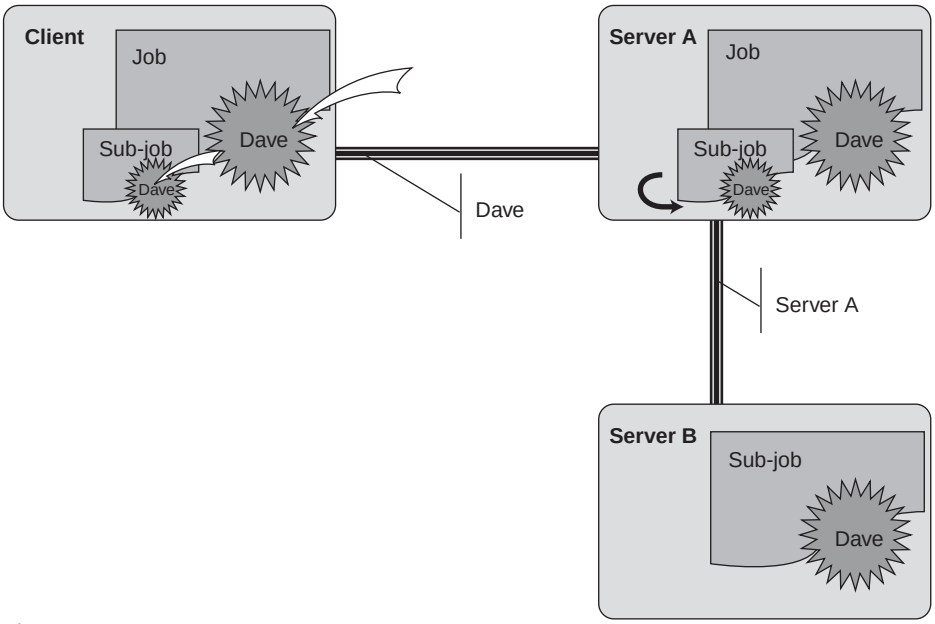

Figure 2
UNICORE V4 multiple-site job from UNICORE client.[note 4)]

note 4) Servers are also allowed to retrieve the results of jobs they have consigned and to delete jobs they have consigned.

job will take, including the resources required to execute each step, before the job is submitted. This description is signed, and the UNICORE servers execute this description if it has not been changed since the signing. Although the description allows some flexibility, for example, conditional execution of parts of the job and a generic request to run anywhere suitable, there is insufficient flexibility to adapt to changing circumstances as required by some Grid applications.[note 5)]

An essential part of the UNICORE approach is the separation of authentication from authorization. Certificates are used solely for authentication and carry no authorization information. Authorization of actions is contained within the signed AJO, and authorization of users within sites is performed by the server using site local rules.

There is no need in UNICORE to contact any other parties to create a delegated job; endorsing is an activity carried out entirely on the client side.

2.4 Grid security infrastructure (GSI)

The GSI from the Globus Project achieves delegation using "proxy certificates".[2)] A proxy certificate is an X.509 certificate issued by the end-user (not a CA) to a process acting on the end-user's behalf. The proxy certificate carries the identity of the end-user. That is, it can be used to establish SSL connections and is interpreted by sites using the GSI as authority to perform work on the end-user's behalf. An entity presenting a proxy certificate within the GSI will be granted all the rights of the end-user.

Proxy certificate holders can issue proxy certificates to other entities, which makes dynamic delegation very easy within the GSI because the details of the execution do not need to be known in advance. Furthermore, as the right to delegate can also be passed on, the resources used during the execution can also be dynamic.

Simple implementations of the GSI proxy mechanism allow full impersonation rights; the holder of a proxy certificate gains all the rights of the user at a site. Version 3.2 of Globus Toolkit[2)] introduces a VO-like Community Authentication Server (CAS) that issues certificates marked up to restrict the rights of the bearer to those agreed between the VO and the resource owner. Impersonated rights can be restricted in this way, but as the agreement does not involve the user, these rights will be broader than the minimum required to perform any particular task.

The private key bound to a GSI proxy certificate cannot be stored in an encrypted form. This is because it must be read by the processes to which the rights have been delegated without contact with the end-user. Therefore, for example, direct authorization of access to the private key is not possible. The GSI proxy private key is typically stored unencrypted and protected only through the normal file system mechanisms of the site. This means that, for example, any systems administrator at a site can read a GSI proxy identity.

The combination of impersonation rights and unencrypted storage means that GSI proxy identities are both valuable and relatively easy to acquire. The main mechanism for limiting the damage caused by unauthorised acquisition of a GSI proxy identity is to limit the lifetime of the delegation. GSI proxy certificates have typical lifetimes of the order of hours or days.[note 6)]

The relatively short lifetime of GSI proxy certificates causes problems for long-running processes because the proxy can expire before the process has completed. Therefore, a mechanism for extending the lifetime of delegated rights is required. Condor-G, for example, overcomes this by emailing end-users when they need to refresh

note 5) The AJO endorsement applies to the tasks to be performed, but does not control the contents of files. There is therefore some inter-site trust necessary for file transfer. The receiving site has authorization from the end-user to write the file, but if the send site has been compromised, the contents may not necessarily be what the end-user expects.

note 6) This seems to be the only way to invalidate GSI proxy certificates. There seems to be no revocation mechanism.

their delegations; however, this defeats the purpose of the GSI proxy by requiring end-user intervention with a delegated process, for example, by falling back on direct authorization. On the other hand, any scheme to automatically refresh delegation without end-user intervention would run the danger of creating an indefinitely prolonged delegation.

The GSI imposes a transitive trust requirement on all sites participating in a particular Grid (members of a particular VO). Because possession of a proxy certificate alone is sufficient to authorise work at remote sites, any site accepting GSI delegations must trust that the originating site and all other sites in the delegation chain are properly managed and not compromised.

Delegation within the GSI requires the involvement of at least the end-user and the server and may require three parties if a CAS is used. The delegation protocol needs to be carefully designed because the entities issuing a proxy certificate (e.g., end-users and proxy authorized processes) must be careful to authenticate the identity of the process requesting the proxy so that delegated rights are given to the correct process.

3. Explicit Trust Delegation

We have described the two main approaches to delegation within Grids: that used by UNICORE V4 and that used in the GSI. These two approaches can be considered to be at different ends of a spectrum that ranges from inflexible but secure to flexible with insecurities. The most desirable delegation mechanism would be placed near the middle of this spectrum, combining the flexibility of the GSI with the security of UNICORE V4. The proposal in this section attempts to achieve this balance.

As mentioned in Section 2.3, when AJOs are consigned between end-user and server, only two roles are involved: consignor and endorser. In Figure 1, end-user Y is both consignor and endorser: end-user Y first endorses an AJO and then consigns it to a server. In the case of sub-AJOs (Figure 2), the end-user keeps the endorser role, but Server A becomes a consignor to Server B.

In the UNICORE V4 model, the Endorser is used for two purposes: to authorize the actions (by signing the AJO) and identify on whose behalf the action is to be performed. This means that a set of actions can only be created by the person who wishes to execute them; that is, a user cannot delegate the ability to create a set of actions. The Explicit Trust Delegation extension to UNICORE adds a new role to the model in order to separate these two purposes. The User role identifies the end-user on whose behalf the action is to be performed. The Endorser role is retained but now only authorizes the actions (still by signing the AJO).

Figures 3 and **4** show how normal UNICORE jobs are managed under the new scheme. Here, the end-user can have any of three different roles: the “User role” to identify the user, the “Endorser role” to authorize the actions, and the “Consignor role” to consign jobs to the server. The significant change from Figures 1 and 2 is that Server A may now play both the Consignor and Endorser role, but under the condition that only the end-user takes on the User role.

UNICORE servers supporting Explicit Trust Delegation will implement the following set of rules to define the behavior of each of these three roles:

1) Only the User role is mapped to a local identity and has tasks executed on the User’s behalf.
2) Only end-users can take on the User role.
3) End-users and UNICORE agents (e.g., servers, portals, resource brokers, schedulers, etc.) may take on the Consignor role.
4) UNICORE agents may take on the Endorser role to authorize actions on behalf of an end-user.
5) All Endorsers other than the User must be entered explicitly as trusted agents allowed to endorse the end-user’s jobs.

During processing of an AJO at a server, the

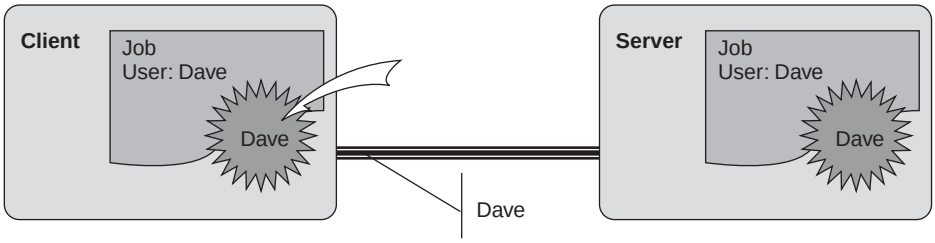

Figure 3
Explicit Trust Delegation framework single-site job.

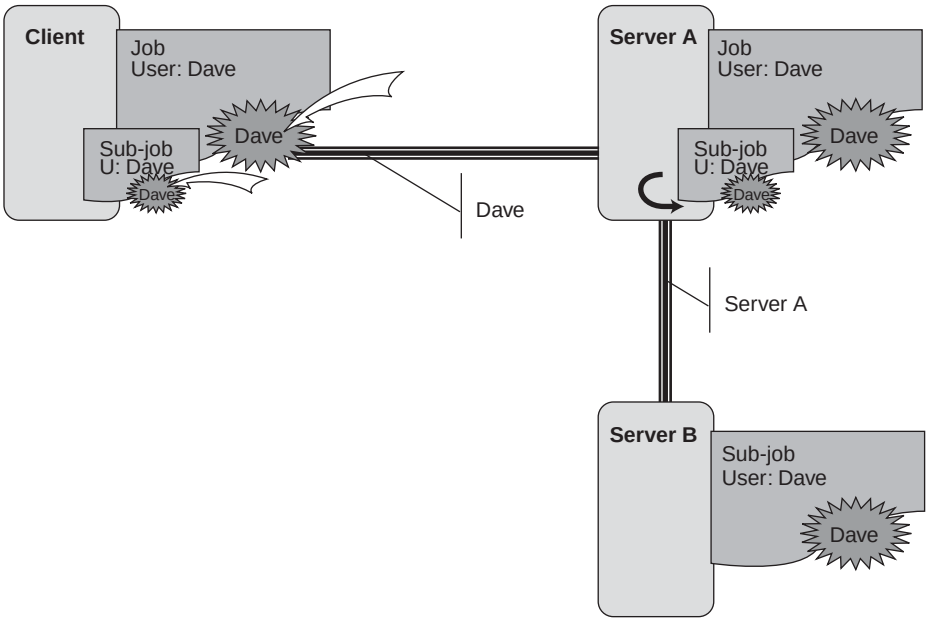

Figure 4
Explicit Trust Delegation framework multiple-site job from UNICORE client.

server authenticates the Consignor and Endorser; ensures that the User has authorized access to the site; and verifies that, if the Endorser is not the User, the Endorser is authorized to endorse on behalf of the User. All this information is coded explicitly into the authorization database maintained at each site.

By way of example, we return to the portal use case (**Figure 5**). The portal can now be granted Endorser rights at sites that explicitly trust that portal to endorse AJOs on behalf of known end-users. An end-user may contact the portal, for example, by using https. Based on the https connection, the end-user's client can authenticate the portal as a trusted agent in the Grid. The portal takes input from the user to construct an AJO, which will include a copy of the end-user's certificate (typically obtained from the https connection) in the User role. The portal then endorses the AJO and consigns it to a server of choice, for example, the portal's certificate server, as both Consignor and Endorser. The server then checks that the User is allowed to run the task at the site and that the Endorser is explicitly trusted to endorse on behalf of the User.

Notice that if this AJO contained sub-AJOs intended for other sites, the sub-AJO would be consigned to the secondary site with the primary

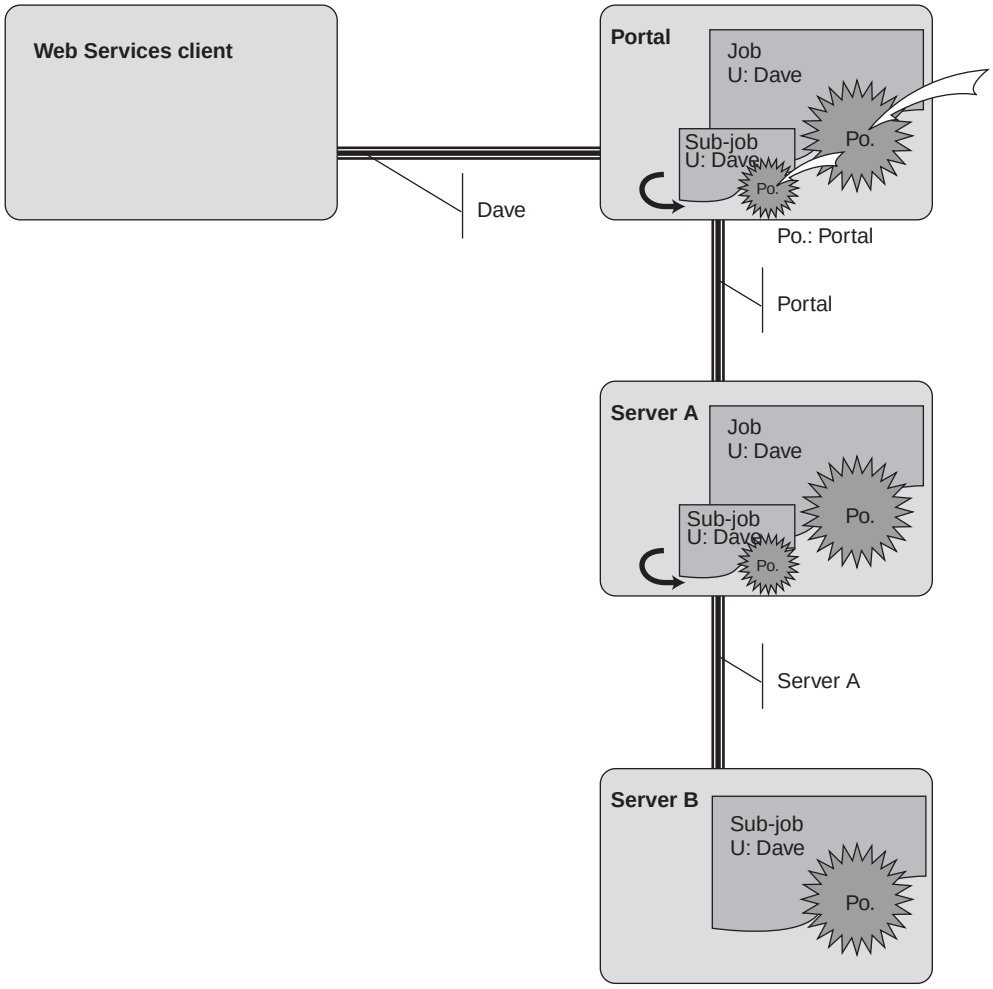

Figure 5
Explicit Trust Delegation framework multiple-site job from Web Services client to explicitly trusted portal.

site in the Consignor role, the portal in the Endorser role, and the end-user in the User role. The secondary site would need to be configured to accept AJOs from the Consignor site, accept the User, and explicitly trust the portal as the Endorser.

The introduction of the User role into the UNICORE model allows dynamic delegation because a set of actions can be created by a third party on behalf of an end-user without the end-user needing to explicitly authorise the actions.

4. Related work

We have investigated other security models such as Kerberos[3] and DCE[4] for their ability to allow dynamic Grids.

Kerberos uses conventional (i.e., not public key) cryptography technology to provide authentication. A Kerberos server provides a trusted third-party service. An authentication server (Key Distribution Center) sits between client and server to issue credentials when a client requests them. When a client sends a request, the authentication server replies with credentials encrypted in the client's key. These credentials consist of a "ticket" for the server, which is encrypted in a server's key, and a session key as a shared secret key for both client and server. Each ticket contains the client's ID and a copy of the session key. The client can then transmit the ticket to the server.

Kerberos 5 was used as the basis for the GSI design[5] and so has many similarities to GSI, allowing delegation through "Ticket Granting Tickets," which can be both forwarded and used to create further tickets.[4] As such, it has many of the advantages and some of the disadvantages of the GSI. However, it should be noted that the creation (and delegation) of a ticket requires the involvement of a third party; namely, the Kerberos server. In contrast, GSI delegates with just the involvement of the User and Server and UNICORE can create a delegation with a single party (the Endorser). Furthermore, Kerberos authorization requires a user connection (i.e., direct authorization rather than delegation).

The DCE (Distributed Computing Environment) security model is a superset of Kerberos; it allows a server to permit either authenticated or unauthenticated access. If a client sends an authentication request to a server, the server replies with just the service it uses, which can be none or DCE secret key. If both parties agree to the condition, the client requests authentication and the server provides the service. Authentication is carried out at runtime as part of the Remote Procedure Call (RPC) protocol.

DCE authorization is direct; if the user (client) disconnects from a DCE session, the authentication becomes invalid.

5. Future work

Administering authorization databases in large or very dynamic Grids will be quite a burden if every user, Endorser, and server needs to be added individually. The infrastructure of a VO takes responsibility for identifying these entities, for example, by issuing X.509 certificates. We plan to investigate ways of automating this process and will consider the implications of allowing the VO to decide the trust levels for the whole organisation. For example, it could be possible to allow endorsing if both the User and Endorser are in the same VO. The extent to which this flexibility can be supported in a standard way will depend on extensions to existing standards in authentication and authorization.[6),7)]

Using a PKI as the basis for authentication and delegation has the advantage that the end-user and server can communicate directly with each other without the need for a trusted third party. However, one disadvantage is that users must always have access to their private keys, which may be difficult if they use more than one device to communicate with the Grid.[note 7)] One solution to this is to introduce a trusted third party with whom the user can lodge credentials and that can be accessed by servers (for an example, see the MyProxy servers developed using the GSI).[8)] We will investigate how these can be integrated into the Explicit Trust Delegation framework, for example, by users lodging their willingness for a certain Endorser to create jobs on their behalf with such an authorization server.

As with most Grid systems, UNICORE is moving towards Web Services.[9)] This means that the delegation mechanisms will need to be adapted to work within existing Web Services infrastructures. There are also Web Services technologies that address aspects of the delegation problem, for example, WS-Security.[10)] We will investigate how these can be applied to the Explicit Trust Delegation framework.

It may be necessary to allow some level of end-user authorization of the creation of a particular AJO by an Endorser. To do this, the User will need to insert some sort of token stating that the User has allowed the Endorser to create an AJO. We will investigate ways of creating this token, which could include the User signing some pieces of information, for example, the Endorser's certificate, an Identifier of the particular AJO being created, and/or a time range for which the delegation is allowed. Adding this extra authorization step will reduce the flexibility of the system,

note 7) Storing a private key on a portable device such as a smart card is not a viable option at the moment as the card readers are not widely available.

so the first step will be to establish whether it is necessary.

6. Conclusions

Grid jobs combine a number of sites into a single workflow. If this is to be allowed, then the administration of the sites in a Grid must trust the other sites in order to accept work requests from them. The degree of trust that is required differs between the UNICORE and GSI models.

Explicit Trust Delegation is an extension of the UNICORE architectural model that allows trusted agents in a Grid to create AJOs (effectively, Grid actions) on behalf of end-users. The trust granted to these agents is made explicit in the model, thus allowing site administrators the flexibility to manage the trust relationships in the Grids to which they belong.

It is the authors' belief that these ideas could apply equally to other Grid systems based on PKI and certificate mechanisms.

References

1) I. Foster et al.: The Anatomy of the Grid, Grid Computing: Making the Global infrastructure a Reality. Berman, Fox, and Hey editors, 2003.
2) V. Welch et al.: Security for Grid Services. Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), Seattle, June 22-24, 2003.
http://www.globus.org/Security/GSI3/GT3-Security-HPDC.pdf
3) B.C. Neuman: Proxy-Based Authorization and Accounting for Distributed Systems. Proceedings of the 13th International Conference on Distributed Computing Systems, May 1993, p.283-291.
4) DCE 1.1: Authentication and Security Services. Open Group CAE Specification Document Number C311, October 1997.
http://www.opengroup.org/onlinepubs/9668899/front.htm
5) V. Welch et al.: X.509 Proxy Certificates for Dynamic Delegation. 3rd Annual PKI R&D Workshop, Gaithersburg, MD, 2004,
http://www.globus.org/Security/papers/pki04-welch-proxy-cert-final.pdf
6) Security Assertion Markup Language (SAML) V1.0. OASIS 200205, November 2002.
http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip
7) Extensible Access Control Markup Language (XACML) V1.0. OASIS Standard 200301, February 2003.
http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf
8) J. Novotny et al.: An Online Credential Repository for the Grid. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
9) P. Wieder et al.: Creating Interoperability between Grids. Cracow 03 Grid Workshop, Cracow, October 27-29, 2003.
http://www.cyfronet.krakow.pl/cgw03/presentations/w-3.pdf
10) Web Services Security: SOAP Message Security 1.0. OASIS Standard 200401, March 2004.
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Dr. David F. Snelling received the Ph.D. in Computer Architecture from the University of Manchester, Manchester, UK in 1993 while working as a research Fellow and Lecturer. He joined Fujitsu European Centre for Information Technology in 1997 and transferred to Fujitsu Laboratories of Europe Ltd., Middlesex, UK in 2002, where he worked in computer systems design and Grid computing. He is responsible for the architecture and development of UNICORE. Dr. Snelling is currently Area Director (Architecture) of the GGF, co-chair of the OGSI Working Group and of the WSRF Working Group in OASIS. He is also chair of the EU's Next Generation Grids Experts Group and serves as an advisor to several organizations on the issues of Grid computing.

E-mail: david.snalling@uk.fujitsu.com

Sven van den Berghe received the B.Sc. (Honors, 1st Class) in Mathematical Physics from Nottingham University, Nottingham, UK in 1979. He continued his studies at Nottingham University on a University Scholarship, receiving a Ph.D. for work in Computational Chemistry in 1983. He joined Fujitsu Ltd., London, UK in 1997, working first for Fujitsu European Centre for Information Technology and then transferring to Fujitsu Laboratories of Europe Ltd., Middlesex, UK in 2002. During his time with Fujitsu, he has been working on Grid computing, primarily in the development of UNICORE.

E-mail: sven.vandenberghe@uk.fujitsu.com

Vivian Qian Li received the M.Sc. degree in Computing Science from Staffordshire University, Staffordshire, UK in 2001. Her Ph.D. research started from July 2001 and mainly focuses on dynamic creation of ASP applications using the meta-modeling approach. She joined Fujitsu Laboratories of Europe Ltd., Middlesex, UK in January 2004, where she has been engaged in research and development of Grid computing for many EU Grid projects, including GRIP (Grid Interoperability Project), EUROGRID, UniGrids, and NextGRID.

E-mail: vivian.li@uk.fujitsu.com