Programming Research Group

ARCHITECTURES FOR SECURE DELEGATION WITHIN GRIDS

Philippa J. Broadfoot and Gavin Lowe

PRG-RR-03-19

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

The purpose of a *grid* is to enable large scale distributed computing over wide area networks, where entities (for example, users) can gain seamless access to computing resources across heterogeneous and geographically dispersed environments. There are a number of difficult issues that arise within the design and deployment of such a grid architecture; security has been a particularly difficult issue. In this paper, we will focus on the security implications arising through the introduction of delegation, an essential requirement to enable the sort of distributed collaboration and resource sharing for which the grid is designed. The precise impact of proposed delegation mechanisms upon security remains unclear within many grid projects. What security guarantees are required from the delegation architecture, and how does one determine whether a particular architecture provides those guarantees? In this paper, we aim to address these issues more precisely. We begin by identifying a number of security aspects of delegation. We then consider two existing architectures for secure delegation and evaluate what security requirements they meet. Finally, we discuss their applicability in practice within a grid environment, focusing mainly upon our observations within the European Union DataGrid project [Dat].

1 Introduction

The purpose of a *grid* is to enable large scale distributed computing over wide area networks, where entities (for example, users) can gain seamless access to computing resources across heterogeneous and geographically dispersed environments. Thus, users of a Grid will be abstracted away from the physical distribution of available resources and, instead, will be presented with a single virtual environment. A *computational grid* is concerned with computationally intensive applications and enables the sharing of distributed computational resources, for example, distributed supercomputing. A *data grid* is typically concerned with applications of a data intensive nature and therefore seeks to enable the sharing of distributed data resources, for example, data mining across distributed databases.

Over the last few years, numerous research and industrial projects have emerged with the aim of developing grid technology, for example: Globus [Glo], European Union DataGrid [Dat], IBM IntraGrid [IBM], CrossGrid [Cro] and GridPP [Gri]. The applications for which these grids are being developed are equally diverse, ranging from computationally intensive physics experiments through to the development of advanced methods of breast cancer screening and diagnosis.

In this paper, we will focus primarily on our observations within the EU DataGrid project. This is a 3 year project (starting in 2001) involving 21 partners in 15 countries. Their objective is to build a computing infrastructure that will enable intensive computation and access of large-scale shared distributed databases, from hundreds of TeraBytes to PetaBytes, across widely dispersed communities. The targeted application areas include Earth Observation, High Energy Physics and Bioinformatics, from which security requirements have been derived.

There are a number of difficult issues that arise within the design and deployment of such a grid architecture, for example, scalability, efficiency and ensuring varying security requirements across heterogeneous environments. Security has been particularly problematic for a variety of reasons, not least because it was not incorporated into the overall design from the start of the project. Furthermore, the security requirements for each application area vary enormously, ranging from some of the physics experiments, which require basic levels of security such as authentication and integrity, to the biomedical applications which require much more stringent security in both the storage and transport of medical data.

In this paper, we will focus on the security implications arising through the introduction of delegation of users' rights. Delegation is essential to enable the sort of distributed collaboration and resource sharing for which the grid is designed.

An important requirement in a grid is *single sign-on*, defined in [FKTT98] as: "A user should be able to authenticate once (for example, when starting a computation) and initiate computations that acquire resources, use resources, release resources, and communicate internally, without further authentication of the user." The idea is that a user can have access to a large number of distributed resources without having to actively authenticate himself to each one individually. A single job may involve many users and resources with computations running in parallel (as opposed to the traditional one-to-one client to server interactions). In addition, users need to be able to set computations running for long durations of time (for example, several weeks) without necessarily remaining logged on to their local system throughout. Furthermore, a job submitted by a user U to some resource R may not necessarily only involve R; R may need to access other remote resources on U's behalf to successfully complete the job. For this type of work, some sort of delegation is required, while maintaining security for both users and resources.

Delegation can be achieved, for example, by a user U signing appropriate credentials that specify the delegated rights, and passing them to the delegate A; A presents these credentials in order to exercise those rights. The aspect that makes delegation interesting is that we need to be sure that A—or anybody else—cannot misuse those delegation credentials. As we will see, there are several subtle questions that arise.

Although delegation is conceptually simple, designing and deploying it within a grid environment has proved to introduce problems regarding security. The precise impact of proposed delegation mechanisms upon security remains unclear within the project. How does one determine what level of security is guaranteed when deploying a particular delegation mechanism? We aim to addresses these issues more precisely in this paper.

A few words about terminology: we will use the word *agent* for an arbitrary entity able to take part in protocols; we will use the word *delegator* for an agent that delegates rights and the word *delegate* for the agent to whom those rights are delegated.

In the next section, we consider what security aspects arise when considering delegation architectures, and highlight some problems that might occur. In Sections 3 and 4 we consider two existing delegation architectures, and analyse what security requirements they satisfy when considering the security aspects identified in Section 2.

2 Security aspects of delegation

In this section, we outline what we believe are the main security aspects that need to be considered when analysing the level of security provided by a given delegation architecture and its suitability for a particular application domain.

Restricted delegation For a delegation scheme to be of practical use, it must be possible to enable restricted delegation: A delegator must be able to determine precisely the level of their rights to be delegated at any time. For example, a user U may lodge a simulation program and associated data on a resource. U may then decide that it is acceptable for another agent A to use the simulation program and obtain the corresponding results, but that A should not be able to access the raw data.

In some existing schemes, it is common for delegation to be implemented as an "all or nothing" option. We take the view that, unless the delegation mechanism is deployed in a fully trustworthy network, this is not acceptable: personally, while we might be willing to delegate some of our rights, we would certainly not want to delegate all of our rights to an agent that we did not fully trust. We similarly take the view that onwards delegation should not be possible without the permission of the original user.

Authentication The authentication requirements between two nodes within a grid infrastructure are not particularly distinct from those traditionally experienced in client-server networks. However, when delegation is enabled (especially onwards delegation), authentication no longer only involves two nodes and precise nature of what authentication is required between nodes becomes unclear. Before deploying a delegation mechanism, one needs to be clear as to what authentication relationships are established between participating nodes. For example, should each step of a delegation chain be authenticated? If so, then who should be authenticated to whom? Should every agent in the chain be authenticated to the user and/or to the resource? Authentication is a well researched area in security; however, descriptions of existing delegation schemes are surprisingly vague on this question.

A related question, in the common case where the authentication mechanism makes use of public key certificates, concerns the trust given to different certification authorities (CAs). It might be that different agents in a delegation chain trust different CAs. Under what circumstances, then, should a certificate signed by a particular CA be accepted? For example, the European Union DataGrid Project [Dat] uses 13 different CAs, and has 13 different virtual organisations; it is not expected that every virtual organisation will necessarily trust every CA; further, some virtual organisations might accept certificates from CAs outside the DataGrid project.

We take the view that the CA should, at least, be trusted by the original user, whose rights are being delegated. It is also reasonable for each agent in the chain, particularly the end resource, to only accept certificates signed by CAs it trusts.

Integrity When a user U delegates a set of rights to some agent A (for example, by signing a credential), the delegation mechanism must ensure that the proof of these rights cannot be modified. It should not be possible for A to be able to perform actions that were not granted in the delegation agreement and that A was unable to perform, before the delegation of rights.

Trust When delegation is employed, one needs to be clear about the level of trust one is assuming between the different components. It is clear that a user U should delegate rights to some agent A_1 only if U trusts A_1 with those rights, and the delegation mechanism will not be able to protect U if A_1 proves to be untrustworthy. However, does this trust encompass the choices of who A_1 trusts? For example, can A_1 decide to pass U's rights on to another agent A_2 that A_1 trusts? Whether or not one chooses to allow this type of *onwards delegation*, the delegation mechanism needs to state these trust assumptions clearly. In practise, a delegating agent should have full control over who it chooses as trustworthy; in the case of onwards delegation, it should have control over what rights can be forwarded and to whom. For example, it should not be possible for an agent A_1 who is acting on behalf of a user U to forward U's rights onto a third party A_2 without U's permission.

Revocation For a delegation scheme to be practical, there needs to be an effective revocation mechanism, to revoke delegation credentials, where a delegate has been compromised, where a delegation key is compromised, or where the user decides he can no longer trust the delegate.

In the case where onwards delegation is allowed, if the credentials given to A are to be revoked, it will often be required to revoke the credentials of all agents that A has delegated on to, and transitively onwards.

Freshness The delegation mechanism must ensure that delegated rights are used freshly: that they are (i) used only within the time-frame allowed by the user and (ii) not used more times than allowed by the user.

Timeliness can be achieved by the use of a suitable lifetime within delegation credentials. However, as pointed out by Gong [Gon92], the use of timestamps requires the clocks of the different agents to be loosely synchronised; further, this makes the time-maintenance mechanism everywhere part of the security architecture. For this reason, timestamps are not ideal. Alternatively, timeliness can be achieved via a nonce exchange.

The delegation mechanism must ensure that delegated rights cannot be used more often than allowed by the user. For example, user U might delegate to some agent the ability to access a particular resource precisely once if each access costs U money. The delegation mechanism must enforce this requirement.

Responsibility If an agent A accesses a resource R on behalf of some user U, who is responsible for the subsequent actions performed upon R? Certainly, both U's and A's identities should be recorded in audit logs. Further, the delegation credentials linking A to U should be stored, and these credentials should satisfy the property of non-repudiation. This is not a question we consider further in this paper.

This list of points is not exhaustive, but highlights some of the main issues that we believe need to be considered when implementing delegation schemes (in addition to the standard network security requirements).

In the following two sections, we present two existing delegation mechanisms that differ significantly from one another. We discuss their applicability in practise within a grid environment, with regards to the security issues we have discussed above. The first scheme was proposed by Gasser and McDermott [GM90] and is very much favoured by grid developers, in particular within the DataGrid project, due to its delegation chaining mechanism. The second scheme was proposed by Foster *et al.* [FKTT98] with the specific intention of being used in a grid environment. Despite this, these protocols are not a popular choice within the DataGrid project.

3 Scheme I: Delegation chaining

Gasser and McDermott propose a delegation mechanism in [GM90]. The aim of the scheme is to enable a resource R to verify whether an agent A, requesting access on behalf of some user U, has been authorised by U, without having to communicate directly with U. This is achieved through the use of delegation credentials[1] that are dynamically created at each step. Digital signatures are used to ensure integrity of the credentials.

The basic delegation scheme works as follows, where U is delegating rights to some agent A:

1. User U and A mutually authenticate themselves in some way, for example via a long-term public-key certificate and using an SSL handshake.

2. The delegate A creates a *delegation key pair*, which consists of a temporary public/private key pair, pk_A and sk_A respectively. The public component pk_A is sent to the delegating user U. The private component sk_A is kept secret within A.

3. U creates a *delegation credential* containing A's identity, the temporary public key pk_A, U's identity, and a lifetime, all signed with U's secret key. U sends the delegation credential, together with U's certificate, to A.

4. A can access resource R on behalf of U by sending the delegation credential and U's certificate to R. R verifies the certificate and credential. A then proves knowledge of the corresponding secret key sk_A via some suitable challenge-response; for example, if the credential is constructed to be an extended X.509 certificate, then it can be used as part of an SSL authentication handshake. R then checks that U is himself allowed to access the resource, and if everything is in order, allows A access.

The above scheme can be extended to allow onwards delegation. If U can delegated some rights to A_1, and A_1 wants to delegate those rights onwards to A_2, then A_2 creates a temporary public/private key pair, pk_{A_2} and sk_{A_2}, and sends the public key to A_1; A_1

[1] Gasser and McDermott refer to these as delegation *certificates*, although we prefer to reserve the word "certificate" for long-term public-key certificates.

creates a delegation credential for A_2, including pk_{A_2}, and signed using his own temporary secret key sk_{A_1}; A_1 then sends this new credential to A_2, along with his own credentials.

This can be extended further to a delegation chain $A_1, A_2, \ldots, A_n$, where each agent delegates to the next, creating a suitable delegation credential. When A_n wants to access a resource R, it passes the entire chain of credentials. R verifies that all agents in the delegation chain are authorised delegates for user U (for example, via an access control list) and that each delegation certificate is authentic (through logical deductions of the chain of credentials leading back to U).

3.1 Appraisal

In this section, we consider what requirements this scheme satisfies, based on the security aspects raised in Section 2. Based on our observations here, we discuss its suitability within a grid environment in Section 5.

Restricted delegation If restricted delegation is required, then the delegation policy can be specified in the delegation credential. We are not enthusiastic about this solution, for it raises tricky questions about how to specify the delegation policy, particularly if there are multiple resources involved. Note also that the delegation policy would have to specify whether onwards delegation was allowed, which rights could be delegated onwards, and to whom. The problem of the resource checking that all of the delegations are allowed seems difficult and potentially unscalable.

Trust This type of delegation scheme relies on the assumption that there is trust between all the agents within a delegation chain, and that these agents remain trusted (i.e. not compromised) while the delegation chain is in use. Agents are expected to destroy their private delegation keys when the delegation is no longer needed, so as to lessen the damage if the agent is subsequently compromised. How is this ensured in practice? If an agent A in some delegation chain can be compromised after the corresponding secret delegation key has been destroyed, what mechanism is in place to prevent A being compromised beforehand?

This interdomain trust assumption is a very restrictive one when employing this delegation mechanism in a practical environment such as the Internet. As discussed in [GM90], without this assumption the system is rendered insecure.

Revocation Gasser and McDermott's delegation architecture contains minimal support for revocation. They state that once the delegation is no longer required, all delegates are expected to destroy the secret component of their delegation key pair, to protect against subsequent comporomise of any of the agents in the chain. However, this mechanism provides no protection against compromise of an active delegate. Further, the mechanism requires all delegates to be trusted to actually destroy their secret keys when finished.

If delegation credentials are limited to allow the delegate to access a small number of resources, then the user can revoke those credentials be directly accessing those resources in question; this mechanism works less well with unrestricted delegation.

Authentication Gasser and McDermott state that in a delegation chain $U, A_1, \ldots, A_n$, each agent should be authenticated to the previous agent in the chain, through the use of a long-term public key/secret key pair with an appropriate certificate signed by a CA. The precise details of the authentication protocol used are not stated, but we agree that this is an orthogonal issue.

One implication of this concerns the case where different agents in the chain trust different CAs: it might well be the case that some of those agents are authenticated using certificates signed by CAs that are not trusted by U; it is clearly undesirable that U's rights are delegated to agents who could not be authenticated to U. This scenario could be prevented by the list of CAs trusted by U being passed through the chain, in some authenticated manner; however, this relies upon each agent being trusted to respect that list.

Further, the final agent A_n is authenticated to the resource R only through the delegation chain. One interesting aspect of this is that the credential could be constructed to be an extended X.509 certificate, so that it can be used as part of an SSL authentication handshake. However, for this to work, R has to trust all the other agents in the chain to have correctly authenticated the next; further, R has to trust all the CAs signing the different agents' certificates. It would seem straightforward—and more satisfactory—to arrange for A_n to be authenticated to R using A_n's long-term public key/secret key and certificate, thereby separating the issues of authentication and delegation.

Gasser and McDermott do not discuss authentication of each agent in the chain $U, A_1, \ldots, A_n$ to the next, nor authentication of the resource R to A_n, although the need for such authentication is less clearcut. In any case, it would be fairly easy to add on such an authentication, either certifying agents via the delegation chain or via long-term certificates.

This scheme protects the users and resources from replay attacks by enforcing the last agent A_n in a delegation chain (of length n) to prove to the resource R it is trying to access, that it holds the corresponding secret key of its delegation key pair. This is achieved through some challenge/response between A_n and R. Under the interdomain trust assumption and assuming a "secure" challenge/response protocol between A_n and R, a malicious observer cannot simply gather a copy of all the certificates chains and replay it to R, since it would not be able to prove possession of the secret key.

Freshness The delegation credentials contain a lifetime, designed to ensure that rights are exercised only within the timescales allowed by the user. However, this suffers from the problem of requiring synchronised clocks, discussed in Section 2.

If the user wants to impose some limitation on the number of times a right may be exercised, then this information can be included in the delegation certificate. In order for this to be effective, the resource has to be trusted to keep track of usage, and to respect the policy.

In an economic model, where the user limits usage because the resource charges for each access, then this trust seems reasonable: if the resource allows more accesses that allowed by the policy, then the user will simply refuse to pay for the excess; the delegation credential can be seen as a promise to pay for the usage, and only the usage, specified.

In settings where the user limits access for security reasons, such as to limit access to confidential data, then the trustworthiness of the resource has to be considered on a case-by-case basis.

4 Scheme II: Call-back delegation

In [FKTT98], Foster *et al.* look at some of the security issues that arise in the grid architecture and propose a number of protocols to enable secure inter-domain communications. In this section, we consider three of these: to create a user proxy to act on behalf of a user; to create a process to perform work on behalf of a user; and to delegate to such a process the right to access further resources.

Creating a user proxy A basic form of delegation used within the Grid infrastructure is the creation of a *user proxy*. A user proxy UP is a process created by a user U, to run on behalf of U, on a machine local to U and trusted by U. The aim is to achieve the single sign-on requirement, allowing the user to log in and out of the system irrespective of the duration of the job requested.

Foster *et al.* define a protocol, called Protocol 1, whereby the user supplies the user proxy with a credential C_{UP} containing his own identity, the host of the user proxy, a temporary public key, and a lifetime, all signed by the user's long term public key. The credential can be implemented as an extended X.509 certificate, for example. The user also supplies the user proxy with his own certificate.

Creating processes Protocol 2 of [FKTT98] allows a user proxy to obtain resources so as to run processes. The user proxy communicates with a *resource proxy*, which allocates resources from some pool.

1. The user proxy and the resource proxy authenticate each other using the user proxy credential C_{UP} (supported by the user's certificate) and the resource proxy's long-term public key certificate, respectively.

2. The user proxy sends the resource proxy a signed allocation request.

3. The resource proxy checks that the user is authorised to make the allocation request under the local policy.

4. The resource proxy sends the user proxy information about the resource to be allocated; this is passed securely, making use of step 1.

5. The user proxy verifies the information received in the previous step is as expected, and if so creates a signed *process credential*; the form of the process credential is not made clear, but we presume it to be similar to the credentials described earlier.

6. The user proxy passes the process credential to the resource proxy securely, making use of step 1.

7. The resource proxy allocates the resource and passes the process credential to the new process(es); it is assumed that the resource proxy and resource are in the same trust domain, so this can be done securely.

Processes accessing resources Protocol 3 enables a process P, acting on behalf of a user U with user proxy UP, to obtain some other resource; thus the user proxy delegates rights to the process.

1. The process P and user proxy UP authenticate each other using their credentials C_P and C_{UP} respectively.

2. P sends UP a signed allocation request.

3. If UP decides to honour the request, then it runs Protocol 2 with the resource proxy for the requested resource, as above.

4. UP signs the resulting process handle, and returns it to P.

An important point to note here is that every resource allocation request made by a process has to go through the user proxy.

4.1 Appraisal

The method by which rights are delegated in this scheme by Foster *et al.* (achieved in Protocol 3 above) differs greatly to that of Gasser and McDermott's scheme described in Section 3. Here, delegation of a user U's rights is achieved through a direct call-back between U and the corresponding process requesting to act on U's behalf. The building of a delegation chain underlying Gasser and McDermott's scheme is not possible in these protocols.

We believe that this scheme has several advantages over the delegation chaining mechanism of Gasser and McDermott. In this section, we consider what requirements this scheme satisfies, based on the security aspects raised in Section 2.

Restricted delegation All resource allocation goes through the user proxy, so it is straightforward to ensure that resources are allocated only as consistent with the wishes of the user.

One point to note is that if it is required that the process should have limited access to the resource—for example if the resource is a file store, and the process should have access to only a subset of the files—then this can be easily specified within the allocation request from the user proxy to the resource proxy.

Revocation All resource allocation goes through the user proxy, so the user proxy can refuse to allocate further resources to a process it believes to be compromised. Further, it can keep track of which processes have been given rights to access which resources, so it can cancel a previous resource allocation by directly contacting the relevant resource proxy.

In principle, different processes acting for the same user could perform secure inter-process communication, by authenticating themselves using their process credentials. (Foster *et al.* mention the need for inter-process communication, but do not define a mechanism.) In this case, the user proxy will need to revoke the credentials for compromised processes: this can be achieved by the user proxy broadcasting a revocation message to all processes acting on its behalf (if this can be done reliably), or by each process contacting the user proxy before accepting another's credential.

Authentication Protocol 3 seems to provide authentication between the user proxy and the process in two different ways: the authentication through the credentials in step 1, and the signing of messages 2 and 4; it is not clear why both forms are required. It would seem simpler to either:

- establish a session key as part of step 1, and use this to encrypt the rest of the session (for example, using SSL); or
- to simply sign messages 2 and 4, presenting the credentials if they are not cached.

It is worth noting that in some circumstances it is necessary to sign messages that are included within secure sessions, for example to provide non-repudiation, or to allow the signed message to be passed onto third parties; this does not appear to be the case here.

Similar comments concern Protocol 2, where the user proxy signs the allocation request, in addition to the initial authentication; note, though, that the process credential might need to be signed, even if it is passed within a secure session, for purposes of non-repudiation.

We are reminded of Principle 4 from [AN96]:

> Be clear about why encryption is being done. Encryption is not wholly cheap, and not asking precisely why it is being done can lead to redundancy. Encryption is not synonymous with security, and its improper use can lead to errors.

The resource proxy and the process are authenticated to one another indirectly, via the user proxy. One effect of this is that either might not possess a long-term certificate signed by a CA trusted by the other; this can be overcome by the user proxy passing certificates between the resource proxy and process.

Trust These protocols are designed to work over a public medium such as the Internet and therefore rely on much weaker trust assumptions than was the case in Scheme I. The authentication mechanism is based on the user of certificates signed by Certificate Authorities; therefore, there is an assumption of trust in a pre-agreed set of CAs.

Freshness Freshness (both timeliness and multiplicity) are partially ensured using the authentication handshake at step 1. However, if the signed messages in steps 2 and 4 are independent of that authentication, then they are open to simple replay attacks. This can be prevented by establishing a fresh session key at the start and encrypting accordingly (for example, using SSL).

5 Discussion

Delegation within a grid is essential for achieving large scale distributed computing over wide area networks, where grid entities can gain seamless access to computing resources across heterogeneous and geographically dispersed environments. It does, however, have far greater implications upon security than is frequently acknowledged within many grid projects. In this paper, we have identified some of the main security aspects that need to be considered when evaluating the applicability and effect upon security of deploying a delegation mechanism in practice. We considered two existing (and very different) mechanisms for achieving delegation and discussed how the security aspects are addressed in each case.

We conclude this paper with a discussion regarding the suitability and applicability of these two delegation mechanisms within a grid environment – focusing on our experiences within the DataGrid project in particular – and highlight future research avenues in this area.

5.1 Applicability within a grid

The first scheme by Gasser and McDermott is a popular choice in a number of grid projects, primarily due to its scalability and use of certificates. The DataGrid is designed using Public Key Infrastructure (PKI) and has a network of trusted Certificate Authorities to supply grid entities with certificates. To improve scalability, they enable a grid user U to generate and sign short-term credentials itself. The aims are twofold: Firstly, U is allowed to use its short term credentials to access grid resources, instead of U's long term certificate; because the lifetime of the credential is very short (typically 12 hours versus 1 year for long-term certificates), it is hoped that the effects of a credential compromise will be reduced. Secondly, this can be extended to enable U to sign credentials for other entities, thereby delegating rights. Deploying a delegation scheme like the one proposed by Gasser and McDermott is seen (by some) to be a natural extension to this existing architecture.

However, this scheme is inherently weak in terms of security, even for grid applications that do not have strict security requirements – for example, some of the physics groups require minimal security versus the biomedical groups who have very strict requirements. As made clear in [GM90], the scheme relies upon a very strict trust assumption: All components involved in a delegation chain are assumed to be trustworthy until they have disposed of their respective delegation keys. While this may be a reasonable assumption to make within a small local network, this will not be satisfied by a grid architecture. Furthermore, the dynamic nature of the grid, together with the shear scale, prevents the establishment of inter-resource trust relations beforehand. Without this trust assumption, the system becomes insecure; for example, the assurance of authentication is dependent upon this assumption. Without it, a system within a delegation chain may become compromised before the disposal of its corresponding delegation key. In this case, an intruder can use this key to access resources, supposedly acting on good intentions on behalf of the user, for its own malicious uses.

Finally, there is no effective or scalable revocation mechanism. We believe this to be essential for successful grid technologies.

The second scheme provides a much greater degree of security than the first and would be a better choice for deployment within a grid architecture. Despite the fact that it was developed with grids in mind, it has not been considered in grid projects such as the DataGrid. The main drawback of this scheme is scalability and the fact that all resource allocations go through the user proxy. This could potentially cause the user proxy to become a bottleneck of the system, particularly on large jobs, where many processes are to be acquired.

A final observation we make is that as projects like the DataGrid are becoming more aware of the security issues, they are considering parts or combinations of delegation schemes, in order to balance efficiency requirements with security. For example, the DataGrid is looking towards implementing a restricted version of the delegation chaining model. Restrictions are expected to include: restrictions on the depth of a delegation chain, for example, valid chain can be a maximum length of 2; restricted set of rights delegated (as opposed to the all-or-nothing approach discussed earlier); restrictions regarding what rights can be delegated onwards and to whom. Further details regarding current and future security design plans within the DataGrid project can be found in [Dat03].

5.2 Future research avenues

We would like to develop techniques for formal analysis of delegation mechanisms, especially within the context of grid architectures. We intend to extend the CSP/FDR model-checking approach [LR97, RSG$^+$01] to encompass delegation. Equally, it would be possible to extend other existing analysis techniques based on the Dolev-Yao model, such as those based upon other model-checking approaches [Mea96, MMS97, MCJ97], the Strand Space approach [THG99], the Spi Calculus [AG97], or theorem proving approaches [Pau98, Coh00].

Some requirements of delegation architectures would appear reasonably straightforward to analyse. One would like to be able to show that processes have access to resources only as allowed by the corresponding users; this can be captured by an assertion about the states of different agents, and so can be easily modelled using any of the above approaches. The question of authentication is now well understood, and many analysis techniques can deal with it. To consider revocation, one would like to be able to show that if a right is revoked, then it can no longer be exercised. These analyses will probably reveal trust assumptions: other requirements will be found not to hold when one agent is untrustworthy.

Some delegation mechanisms—and, indeed, other security services—are layered on top of a suitable authentication protocol, such as SSL [Net96, Tho00], which is designed to protect against dishonest outsiders; the delegation mechanism aims to protect against dishonest insiders, once they have been authenticated. In a formal analysis, one would want to abstract away from the use of SSL, modelling simply the service it provides to the delegation mechanism; proving the correctness of such an abstraction is the subject of current research (see [BL03] for further details).

Acknowledgements

This research was supported by funding from the UK Engineering and Physical Sciences Research Council under the project *Security Aspects of the EU-DataGrid*. We would like to thank our colleagues on the DataGrid project for useful discussions about delegation.

References

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the 4th ACM Conference on Computer and Communications Security*, 1997.

[AN96] Martín Abadi and Roger Needham. Prudent engineering practice for cryptographic protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996. Also in Proceedings of the 1994 IEEE Symposium on Security and Privacy, and Digital Equipment Corporation Systems Research Center, Research Report 125.

[BL03] P. J. Broadfoot and G. Lowe. On distributed security transactions that use secure transport protocols. In *16th IEEE Computer Security Foundations Workshop*, 2003.

[Coh00] Ernie Cohen. Taps: A first-order verifier for cryptographic protocols. In *Proceedings of 13th IEEE Computer Security Foundations Workshop*, pages 144–158, 2000.

[Cro] *The CrossGrid Project.* See `http://www.crossgrid.org/`.

[Dat] *The European Union DataGrid Project.* See `http://www.eu-datagrid.org`.

[Dat03] DataGrid Security Co-ordination Group. *Security Design (Deliverable 7.6)*, 2003.

[FKTT98] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A security architecture for computational grids. In *5^{th} ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[Glo] *The Globus Project.* See `http://www.globus.org`.

[GM90] M. Gasser and E. McDermott. An architecture for practical delegation in a distributed system. *IEEE Symposium on Research in Security and Privacy*, pages 20–30, May 1990.

[Gon92] Li Gong. A security risk of depending on synchronized clocks. *ACM Operating Systems Review*, 26(1):49–53, 1992.

[Gri] *The GridPP Project.* See `http://www.gridpp.ac.uk/`.

[IBM] *The IBM IntraGrid Project.* See `http://www-1.ibm.com/grid/`.

[LR97] Gavin Lowe and Bill Roscoe. Using CSP to detect errors in the TMN protocol. *IEEE Transactions on Software Engineering*, 23(10):659–669, 1997.

[MCJ97] W. Marrero, E. Clarke, and S. Jha. A model checker for authentication protocols. In *Proceedings of DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[Mea96] Catherine Meadows. The NRL Protocol Analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.

[MMS97] John C. Mitchell, Mark Mitchell, and Ulrich Stern. Automated analysis of cryptographic protocols using Murϕ. In *IEEE Symposium on Security and Privacy*, 1997.

[Net96] Netscape Communications Corporation. *The SSL Protocol Version 3.0*, 1996.

[Pau98] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[RSG+01] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Pearson Education, 2001.

[THG99] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2, 3):191–230, 1999.

[Tho00] Stephen Thomas. *SSL and TLS Essentials*. Wiley, 2000.