A SECURITY FRAMEWORK FOR DISTRIBUTED BATCH COMPUTING

by

Ian D. Alderman

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2010

For Irene.

ACKNOWLEDGMENTS

I would like to thank my advisor, Miron Livny, for giving me the opportunity to work with the Condor project. The experience of working with a talented, skillful and dedicated team on a large software project has been extremely valuable to me. I am especially grateful for the many times Miron has been supportive and inspiring. The whole Condor team has been a pleasure to work with. In particular I would like to thank Marvin Solomon, Todd Tannenbaum, Zach Miller, Jaime Frey, Peter Couvares, Joe Meehan, Nick Coleman and Will Benton.

I would also like to thank my other thesis committee members: Remzi Arpaci-Dusseau, AnHai Doan, Suman Banerjee, and Kristen Eschenfelder, for their insights, questions, and advice for my research.

In addition to my thesis committee, I would like to thank Wisconsin faculty Mary Vernon and Anuj Desai for giving me interesting problems to work on. The support of Cornell faculty and staff was essential when I was getting started in computer science: Daniel Huttenlocher, Dean Krafft, Fred Schneider, Robbert van Renesse, Lillian Lee and David Gries are fantastic teachers. I also had great teachers in high school: Camilla Titcomb and John Clippinger in particular introduced me to computer science.

Several colleagues at Wisconsin have provided particularly valuable feedback, especially Florentina Popovici, Alexey Loginov, Mike Molla and Hao Wang. Igor Sfiligoi has been an valued collaborator. The support of Jason Stowe and Cycle Computing has been very valuable in the past year and a half. Also, the support and friendship of the Oak Street Ramblers has been invaluable for sanity and balance: May the blood of the Ramblers never run dry.

Finally, I would like to thank my family, without whose love and support this PhD. would have been impossible. My parents have always been there for me when I needed them. Thanks for believing in me! Especially: Irene whose patience and grace has sustained us both, Lillian, who has never known a time when Daddy wasn't "finished," and Miles, who came at just the right time; you three are my inspiration.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

Appendix

Page

Appendix

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

A SECURITY FRAMEWORK FOR DISTRIBUTED BATCH COMPUTING

Ian D. Alderman

Under the supervision of Professor Miron Livny

At the University of Wisconsin-Madison

There is an assumption in the design and implementation of many distributed batch computing systems that once a task enters the system, the system can be fully trusted by all participants, even when the system spans administrative boundaries. As a result, execution hosts and other intermediaries have no way of independently confirming the origin of tasks, attackers have an incentive to attack the intermediaries who handle the tasks, and when results are returned to users, they have no way of determining where and how those results were computed. Users need to be able to specify policies that limit the actions their tasks can perform and the uses to which their delegated credentials can be put, and ways to link these policies to their jobs and credentials.

In this thesis, I address these shortcomings by introducing and analyzing a framework of mechanisms that can be used to reduce the trustworthiness requirements of components in the system. The framework protects execution hosts by making the association between a particular task and a particular user explicit rather than implicit. It protects end users by permitting them to specify a policy regarding task confidentiality and integrity to accompany their tasks. Finally, it protects intermediaries by making them less attractive to attackers. With relaxed trustworthiness requirements on intermediaries, the benefits of sharing tasks and resources between different administrative domains may be realized without relaxing security policies.

Miron Livny

ABSTRACT

There is an assumption in the design and implementation of many distributed batch computing systems that once a task enters the system, the system can be fully trusted by all participants, even when the system spans administrative boundaries. As a result, execution hosts and other intermediaries have no way of independently confirming the origin of tasks, attackers have an incentive to attack the intermediaries who handle the tasks, and when results are returned to users, they have no way of determining where and how those results were computed. Users need to be able to specify policies that limit the actions their tasks can perform and the uses to which their delegated credentials can be put, and ways to link these policies to their jobs and credentials.

In this thesis, I address these shortcomings by introducing and analyzing a framework of mechanisms that can be used to reduce the trustworthiness requirements of components in the system. The framework protects execution hosts by making the association between a particular task and a particular user explicit rather than implicit. It protects end users by permitting them to specify a policy regarding task confidentiality and integrity to accompany their tasks. Finally, it protects intermediaries by making them less attractive to attackers. With relaxed trustworthiness requirements on intermediaries, the benefits of sharing tasks and resources between different administrative domains may be realized without relaxing security policies.

Chapter 1

Introduction

1.1 Overview

Within a community or organization, each member has rights and responsibilities with respect to the other members and the community as a whole. When a community is small, and members know one another, these rights and responsibilities are informal and enforcement is ad hoc. As a community increases in size, and as community members interact with members of other communities, measures to formalize rights and responsibilities and to ensure consistent enforcement can strengthen the community and relationships with other communities.

This is as true for communities formed to share resources used for batch computing as it is for other communities. Within batch computing communities, there are three requirements that must be met for mechanisms to succeed with formalizing rights and responsibilities and ensuring consistent enforcement:

1. The mechanisms must be implemented end-to-end to be effective. When two entities interact, the content of the messages they send each other must be inalterable by and perhaps unreadable to other entities within the community.

2. Users must be able to express policy about where their tasks may be run and how their credentials may be used. Credentials must be issued at the level of tasks rather than solely to users, so that different tasks can have different policies.

3. The mechanisms must not negatively affect the performance of the system.

To achieve the goals of formalizing the rights and responsibilities and enforcing policies, we propose a framework of security mechanisms based on digitally signing messages about the rights and responsibilities of the participants. This thesis describes and analyzes the design and implementation of this framework. The framework consists of a set of security mechanisms that, when used together, can address security concerns for three classes of participants in distributed batch computing: task submitters, resource owners, and infrastructure administrators. For each of these classes of users, the framework may meet more stringent security requirements than present systems permit, and may result in increased cooperation between sites as a result of decreased risks.

The framework is designed to provide end-to-end security: that is, to achieve integrity and confidentiality requirements for the communication between the task submitter and the execution host, and to protect the task from alteration and the user's credentials from misuse. In addition, we permit users who submit jobs to specify policy regarding where their jobs can run and how any accompanying credentials can be used: this policy is enforced by the trusted endpoints and framework-aware resources that are used by the task as it executes remotely. The framework relies on digital signatures: For example, signatures link submitted jobs to the users that submitted them, link credential chains with the jobs that use them, and link jobs with the hosts where they are running. Analysis shows that the performance impact of the security mechanisms is insignificant, and that in some cases performance is improved.

1.2 Motivation

Distributed batch computing systems are often organized into groups: We term a group of users and resources managed by a single entity an *administrative domain.* For example, a particular research project or department within a corporation or university might be an administrative domain. Interoperation between domains occurs between projects or departments. Trust within a domain is high; as the number of interoperating administrative

domains increases, the level of trust between any two domains is likely to decline, reducing the trust each participant has in the system as a whole.

A tension exists in the operation of distributed batch computing systems between the number of participating administrative domains and the trust the participants have of the system. On one hand, an increase in the number of participants can result in increased utilization of existing resources, resulting in greater return on investment in computing resources. On the other hand, number of participants grows, so does the amount of risk involved in participating in the system.

In concert with mechanisms to facilitate interoperation, distributed batch computing systems need security mechanisms that can reduce this tension; mechanisms that can reduce the risks produced by resource sharing between administrative domains. These mechanisms need to address the real risks faced by users and administrators of these systems. In addition, they must be easy to use, easy to adopt, and not affect performance. Finally, every stakeholder who must make changes in order for the mechanisms to be adopted must have an incentive to make those changes.

When the security infrastructures of systems used for distributed batch computing were designed, the submitting user, the infrastructure managing their tasks, and the hosts on which tasks executed were all part of the same administrative domain, and the design of security mechanisms for these systems reflected this.

Over time, features were added to facilitate sharing between administrative domains, and security mechanisms were developed to support these features. Building systems by aggregating the resources of multiple administrative domains was termed "Grid" computing, and a security infrastructure for Grid computing was proposed and adopted [39]. This security infrastructure is the foundation on which our framework of mechanisms is built. Although this infrastructure is an excellent foundation, there are several opportunities to reduce risks and make the infrastructure more responsive to the needs of the stakeholders:

- There are no mechanisms for end-to-end integrity and confidentiality.

- It has no way for users to specify policies about the confidentiality and integrity of their jobs or the use of their credentials, except at a very coarse level.
- It requires that submitting users as well as resource owners have a high degree of trust in the infrastructure components.

Mechanisms for enforcing user-specified policies may be particularly useful when users are paying for compute time (often termed "Cloud" computing, e.g. Amazon's EC2).

1.3 Security Objectives

Our goal is to protect:

1. *Job submitters*:

   The task input and output data: The task itself, the input data to the task, and the output produced by the task should not be alterable (or alteration should be detectable) while the task is in transit. In addition, it should be possible to keep this data secret while in transit.

   Data unrelated to the task: Delegated credentials can be used by the job to read and write data on storage elements and access other resources. Credential delegation should not put data unrelated to the task at risk. Any credentials carried by the task should be usable only on a worker node deemed acceptable by the task submitter, and it should be possible to limit the use of a particular credential to a subset of the resources a user can access.

2. *The operating environment of the worker node*: The worker node and subsequent users need to be protected from the task and attackers who might modify it, and be able to determine the origins of tasks for audit.

3. *The intermediaries*: Because attackers can copy or alter jobs and results and steal credentials by compromising an intermediary, they are attractive targets. If the requirement that intermediaries *must* be trusted can be relaxed, attackers have less incentive to attack them.

1.4 Research Contributions

This dissertation explores ways of addressing the threats facing stakeholders in distributed batch computing systems. As these systems continue to mature and evolve, security mechanisms are required to permit sharing between administrative domains which do not have implicit trust relationships. The thesis that this dissertation supports is the following:

Pervasive use of digital signatures can augment existing mechanisms in distributed batch computing systems to meet security and scalability requirements by reducing the scope of trust assumptions.

The contributions of this thesis include:

- **Analysis of the risks in existing distributed batch computing systems.**

  Commonly used systems, including Condor and GSI, make up an infrastructure for scientific computing that is being used in ways that were infeasible when the systems were designed. We examine the risks of using these systems in the context of trends in distributed batch computing towards increased interoperation between sites and increased adoption of commodity utility computing.

  When a user submits a job, the user must explicitly trust intermediaries that handle their job. This results in risk to the user, the resource that executes their job, and the intermediaries that can become targets of attack.

- **The design and implementation of a framework of security mechanisms.**

  Our framework of security mechanisms provides:

  - Mechanisms permitting credential transformation.

  - Mechanisms to permit end-to-end confidentiality and integrity for jobs, input data and results.
  - A mechanism to link tasks, credentials, and policies.
  - Mechanisms to permit users to specify policies about where jobs can run and how credentials can be used.
  - Mechanisms to enforce user policies.

  Each of the mechanisms address a particular risk present in distributed batch computing systems. Together the mechanisms can permit us to relax the trust assumptions required for operating distributed batch computing systems.

  These mechanisms have been implemented within Condor. Some if not all of the mechanisms will be included in released versions of Condor. The features are easy to use and configure, and users can select from among the features as needed.

- **Analysis of the security and performance consequences of the framework.**

  We analyze the security of the system by assembling a list of security requirements from various sources including those proposed in the context of other frameworks for delegation, and comparing the frameworks with respect to how they meet those requirements. The requirements are described in detail as are the differences between the ways the frameworks meet or fail to meet the requirements. We conclude that our framework meets a wider range of requirements than the frameworks we compare it with.

  We analyze the performance of the framework by comparing the performance of Condor with and without the framework mechanisms, including various combinations of mechanisms and various workloads. We compare the relative overhead of various sizes of job run-time and data size. Finally, we compare the performance of a Condor daemon when processing a large number of messages using authenticated messages vs. authenticated sessions. We conclude that the performance overhead is generally

quite low when reasonable assumptions are made about job characteristics and that performance can actually be better in some cases.

1.5 Organization of the Dissertation

This thesis is organized into three main sections. The first section introduces the framework (this chapter), describes our approach (Chapter 2), and provides necessary background information (Chapter 3). The next section describes the framework in detail, covering the components individually (Chapter 4), how they fit together and within the Condor system (Chapter 5), and how they are implemented (Chapter 6). The third section analyzes the system, beginning with an analysis of framework performance (Chapter 7, continuing with an analysis of security requirements and a comparison with other frameworks (Chapter 8) and notes about related work (Chapter 9). The final chapter (Chapter 10) summarizes our results, describes lessons learned, and discusses future work.

Chapter 2

Approach

2.1 Handwritten Signatures and Digital Signatures

Digital signatures, in the form of signed ClassAds and in the form of certificates, form the foundation of much of the work described in this thesis. This section discusses the meanings and usage of digital signatures in detail.

Imagine if one day you woke up and were unable to sign your name. You would be unable to write checks or use a credit card in person, express your opinions through petitions, form contracts, or, if you were a member of a faculty committee, sign important documents such as dissertation warrants. Credit cards are "not valid unless signed," and signatures appear on currency, driver's licenses, campus identification cards, and contracts. When I was married, and the marriage was not valid until the marriage certificate was signed by the officiant, my wife and me, and witnesses.

However, handwritten signatures are very rarely checked, and very accurate forgeries are possible. A forged check, credit card receipt, petition, contract, warrant or marriage certificate is unlikely to be honored in the long term in the face of repudiation by the signers. As Ueli Maurer points out, the presence of a signature actually corresponds to the awareness of a human performing a "conscious and willful act" [66]. However good the forgeries, my signed dissertation warrant would not long be considered valid without the recollection of my committee that they signed it.

Digital signatures resemble handwritten signatures in many ways, but they also differ in many ways. A technical description of digital signatures and their properties is available in Section 3.4; the key properties are as follows:

1. A digital signature is the cryptographic hash of a piece of data, encrypted with the private (secret) part of a public/private key pair.

2. Digital signatures are easy to produce automatically, but difficult to forge.

3. Given a signature, some data, and a public key, it is very easy easy to determine whether or not the corresponding private key was used with the data to produce the signature.

Analogies are often used to explain technical concepts, with varying degrees of accuracy and appropriateness. It is an easy mistake to make to take the digital signature analogy too seriously, and assume that digital signatures are identical in their properties to handwritten signatures. They are not. Specifically, it is much easier to verify a digital signature than a handwritten one, and much easier to determine whether or not a digitally signed document has been altered.

2.2 Economics, Risks and Threats

Designing secure systems is challenging, in part because there are many techniques available. A key point to remember is that attacks occur because the attacker has some incentive to attack: something they want to obtain by attacking the system. Effective security systems are effective because they change the attackers' incentives. [11, 90]

Consider for example robbing a bank. People would rob banks more frequently if they were not concerned about the consequences of getting caught.

Bank vaults are protected by very strong locks, which take skills and time to break, and they are only opened quickly for customers or employees or customers who authenticate themselves through a variety of methods. Armed guards and/or active alarm systems may

be in place to detect a robbery in progress and prevent the robber from getting in or getting away. Police are quickly summoned, even by automated systems at night. So the clever bank robber realizes that the best chance to get at the money is during the day, when the vault is open and the robbery can proceed quickly. They distract or disable the guard, or find a bank that doesn't have a guard, and pull a gun on the teller. The teller loads all the money into the bag, and the attacker runs away.

Actually, banks employ methods that are aimed at reducing the incentives attackers have of performing this type of attack as well, incentives which continue to function even after the money has left the bank. Tellers and customers can identify the robbers, and cameras are in place in the bank (thus the stereotype of a bank robber with panty hose on their head or a ski mask). Banks can include numbered or marked bills, which may help police to track the robber. Tellers can put a time delay gas and dye bomb in the bag with the money, which may cause the money, and the robber, to be easier to locate.

Most of the work proposed here is similar in theory to this latter category of protection mechanism. If cryptography is like a vault and authentication is like a lock and keys, audit systems are like cameras, marked bills, and dye bombs. The goal is to be able to identify exactly where in the system the attack originated.

2.3 Security Goals and Threats in Distributed Batch Computing

In this section we discuss the particular security goals and threats in distributed batch computing systems.

2.3.1 Distributed Batch Computing

A key insight in the design of secure systems for distributed batch computing is that there are differences between these systems and traditional client-server systems. Specifically, in distributed batch computing, or "grid" systems, several parties participate in computation, parties whose interests may conflict or between whom trust relationships may not exist or may change over time. In many ways, grid systems are similar to the network infrastructure

over which client-server communications occur. However, in addition to network resources, grid systems provide storage and computation resources as well, resulting in different security challenges. Designers of grid systems must be aware of how these differences affect design options and available trade-offs, and make appropriate design decisions.

Designers of grid systems have several goals in addition to security. Security, efficiency, scalability, fault tolerance, and simplicity are all goals of a grid system. Designing a system involves characterizing the trade-offs between each of these goals and trying to find ways that administrators of the system can use the system to suit their particular needs.

A key question for system designers whose systems must run in an untrusted or partially trusted environment is how existing security technologies can be used to address the particular security goals which arise in grid systems. In particular:

1. How can we characterize the difference between grid systems and client–server systems with respect to security?

2. How can we make use of cryptographic and other security techniques used in client-server systems to account for these differences?

3. What other security techniques do we need to incorporate?

To characterize the design goals of a system, first we must understand the risks inherent in the system. A middleware system such as Condor [101] manages grid resources and user tasks which are configured both by administrators and users, interacts with a large number of other systems, and can affect infrastructure at many levels. The security design of such a system needs to include security mechanisms that will protect the interests of users and administrators, including administrators of sites which submit tasks and administrators of sites which perform tasks as well as nonparticipants. This design must be guided by an accurate assessment of what security policies might be desirable, what attacks might occur, and what defenses against these attacks might be practical. In other words, we must understand the threats.

2.3.2 Threats

When we use the word *threats* we mean identification of the attacks most likely to significantly:

A. Violate our security goals, and/or

B. Be likely to occur.

Several details need elaboration. No system can completely eliminate all threats, so the goal in some sense is to identify the worst ones. "Worst" in this context is two-pronged: some threats are unlikely but significant (such as an attacker who is willing to break into a machine room to get access to a cluster), while others are likely but insignificant (such as users who misconfigure their submission scripts to submit millions of useless jobs). The goal is to balance these two: a threat that is significant but unlikely may be worse than one that is moderately significant and moderately likely. In this analysis, we do not attempt to quantify threats. The assignment of likelihoods and consequences is hard to do objectively; fallible human judgment is involved, especially in cases where infrequent events (such as attacks not yet observed) are involved.

Finding threats seems like a simple task, and in some application domains it is, however several authors have touched on it, often either completely outside the realm of computer security [38], or through examples outside it [11, 90]. Risk assessment and economic approaches to computer security have been applied especially among researchers concerned with complex systems [10, 44, 6].

Grid systems are appropriately complex, in part because of the number of stakeholders with differing goals. Since security means different things to different participants in the distributed environments we address, first we must specify the stakeholders and their goals, as well as the goals and strategies that might be used by an attacker to attain those goals. From the perspective of a given stakeholder, however, other stakeholders are potential attackers.

For example, from the perspective of a user submitting compute jobs to the system, a misconfiguration by an administrator of a computing resource which causes files to remain

on the compute resource after task completion might be considered an attack. It certainly allows violation of the confidentiality of the user's data. Similarly, if an attacker A can submit jobs which appear to come from a particular user B, and does so very rapidly, then A can create a denial of service situation in which user B is blamed, and perhaps charged or banned from the system.

The goals and expectations of the users of a deployed system tend to line up with what the system does rather than what it could do. This can cause problems when using the expectations as a guide for adding security mechanisms: since the users don't expect the system to be secure, there's no need to make it secure. The best security systems are ones that allow you to do things you didn't consider before. As a result, we are interested in security goals that may exceed current expectations.

2.3.3 Stakeholders

Stakeholders include:

- **Submitters**, interactive users who submit tasks to the system. In general, their expectations and goals include getting work done quickly, preventing unauthorized users from altering or reading their programs, input, and output. They also suppose that their use of the distributed system will not compromise the security protections they enjoy in the absence of that use.

- **Local administrators** of sites whose users submit jobs. Their primary goal is that use of the grid will not affect the security of their system. Specifically they do not want the mechanisms that are used to allow a remotely running task to access resources that they administer (such as the file servers where their users' data is stored) to be used against them.

- **Resource administrators** of sites that accept jobs. Sites make resources available to outsiders for a variety of reasons, and the administrators of these sites generally want to make sure that their sites continue to be available when the tasks are complete.

They want to make sure that outsiders are adequately sandboxed, that their clusters are not altered by these outsiders. Obviously, they provide the resources so that they can be used, but in addition to providing correct, reliable, and secure resources for customers, their three main concerns are that their cycles and storage resources continue to be available, that their machines are not used for attacks against others, and that the accounting rules are enforced. For completeness, this category also includes administrators of sites which accept jobs, and then re-direct them to other sites.

2.3.3.1 Submitters

The job submitter is primarily concerned with the data security (availability, integrity, and confidentiality) of their input, code, and output. They may be secondarily concerned with the security of audit and accounting data concerning their use of the system, but in general they have no control over this information. In many current grid applications, submitter data confidentiality is not assumed and may not be a high priority, since there's an assumption that in order to get work done on someone else's resources, confidentiality must be sacrificed. However, if some level of confidentiality were provided, users who needed this protection and were otherwise unable to make use of grid resources might be able to do so. For example, the many physicists who use grid resources currently have few confidentiality requirements, but a higher level of confidentiality may be required for pharmaceutical industry researchers.

In many installations, a level of trust (perhaps defined as a financial incentive to cooperate) exists between the resource administrators and the local administrators (and by proxy with the submitters). If such trust exists, it can provide the basis for confidence that the security requirements of the cooperating parties are in fact met. Obviously the strongest trust occurs when the remote administrators and the local administrators are the same, or part of the same management hierarchy. However, the fundamental promise of grid technologies comes from the reduced costs that can occur as a result of increased resource utilization and reduced response time when resources are shared across administrative boundaries. In

many resource sharing arrangements, the resource administrators have some organizational affiliation with the local administrators, such as being part of the same university, or funded on the same grant or through the same agencies, providing a basis for trust.

In some sense, the security problem can be viewed as one of determining responsibility to violations of the security policy. When the same organization manages all the resources that participate in a transaction, and a security policy violation occurs, it's clear that that organization is the one responsible. In a collaboration in which two participants have no basis for trust, neither party has an incentive to respect the security policy of the other. Security systems for collaborations can handle this in two ways: by transforming the computation so that collaborators can not affect the security policy, or by leveraging the existing trust relationships by explicitly determining responsibility.

The same reasoning applies regarding the integrity and availability concerns of submitters, primarily with the output of computation (although this may be extended to input in cases where a pipeline takes output from one computation and uses it as input for another). Some solutions, more efficient than probabilistic checking, exist for the problem of integrity [46, 97]. It is important to note that data integrity compromises can occur as a result of malfunction as well as attack, and that protections against malfunction (such as bits flipped in memory or communications) must be considered as well. Solutions for the prevention or detection of these problems exist (such as error detecting or correcting memory and checksums for packets), but may be inadequate or simply not propagated in a distributed environment.

It is easier to detect violations of data availability, although, again, an important issue is determining responsibility.

2.3.3.2 Local Administrators

Local administrators are also concerned with confidentiality, integrity, and availability, but regarding different data. Attacks of most concern to these administrators are those that affect their infrastructure, including the data of the submitters and other users of the system, as well as the non-data resources they manage. The most significant scenario is that the grid

infrastructure could become the conduit for an attack that extends beyond the compromise of a submitter's account. A poor security design would allow a remotely running task access to the same level of privilege available to the submitter: access to all their files, and the ability to execute code (or write executable files which will subsequently be executed) as that user.

2.3.3.3 Resource Administrators

Remote administrators are responsible for the management of resources used by the submitters. In that capacity, their goal is to make the resources available and effective. These resources generally consist of compute cycles and storage, but also include the grid infrastructure itself. Attacks against this infrastructure may be more significant (for example, a denial of service attack against a machine that matches jobs to available machines may result in no machines being available).

In many cases, resource administrators are also local administrators for groups of users and resources are provided so that other administrators will provide resources in turn. Much of the original design of Condor was for recovery of idle cycles on user desktops during periods when these desktops are not in use. Obviously these desktops have their own security mechanisms in place; however, additional vulnerabilities can be introduced through the addition of grid infrastructure.

Whether or not resource administrators have users of their own to protect, they also are presumed to be able to prevent submitters from attacking one another, or third parties, through their resources. Examples of attacks that might take this form are submitters who are able to prevent other users from obtaining a "fair share" of computational resources, or who are able to manipulate the accounting system in their favor.

2.3.4 Attackers

Another player in the system, who does not really count as a stakeholder, is the attacker. Attacker goals may include the following; no claims are made regarding the relative likelihood (or frequency) of these goals or their significance for a particular organization:

- obtain input / code / output

  As discussed above, many current grid users are not concerned with the confidentiality of their data. However, this does not mean that all potential users of high throughput computing are not interested in confidentiality. Simulations, data mining, biological applications, *etc.* may use sensitive data and/or proprietary code, be easily parallelizable, and could benefit from taking advantage of grid technology. The attractiveness of this goal to an attacker is relative to the sensitivity of the data.

- vandalism / script kiddie

  "Script kiddies" is jargon used by security administrators to describe attackers who have little technical skill other than the ability to download and run an exploit against a system. Generously, these (presumably young) attackers are interested in gaining experience and learning about security. This is a bit like gaining experience as a marksman by playing with guns.

  Although it's easy to joke about this motivation for attack, the ease of it, combined with the number of people on the Internet with little technical skill and questionable judgment, makes it quite likely that sites vulnerable to this type of attack will succumb. Fortunately, grid infrastructure itself is unlikely to be widely distributed enough to excite much attention from this community of attacker.

  Similarly, attackers motivated by the "fame" they get from vandalizing web sites are unlikely to be drawn to grid environments unless they are able to find some aspect of the environment that is visible (like a web page) to deface.

- affect reputation of pool management, ownership, submitter, or deny correct service (grudge).

  A disgruntled former (or current) employee (or student) may be motivated to attack a pool of resources out of revenge, and attempt to do something which would affect the reputation of the owners or management of that pool. If grid resource become a commodity, this could increase as a cause for concern. Attackers may have inside knowledge or access. A denial of service attack against the infrastructure may have the consequence of preventing tasks from being executed. Similarly, a compromised site which frequently returned corrupted data or incorrect results would suffer a loss of reputation, could easily go undetected for long periods of time, and could result in the invalidation of research effort.

  Similarly, an attacker specifically determined to do damage to an individual researcher, a research group, or a sponsoring organization by affecting results, manipulating software so that it behaves badly either within the grid infrastructure (for example submitting jobs at a high rate) or performs attacks against external resources.

  Science has political ramifications; attackers may be motivated to prevent it from being performed. This is particularly true of controversial research, for example, stem cell biology, military simulation. Similarly, an attacker able to return incorrect results might be able to less obviously and more effectively disrupt research.

- obtain CPU time / obtain storage

  Attackers who do not have access to grid infrastructure may be motivated to obtain that access, in order to perform computation (such as cracking captured passwords) or storing data (such as pirated software or media). Clusters with fast machines and large disks might be particular targets.

- use hosts to perform additional attacks

Compromised machines are often used to perform additional attacks; in particular, centrally managed clusters might be efficiently used as part of a denial of service attack.

2.4 Summary

There are several characteristics of grid systems which distinguish them from conventional systems with respect to security, and are particularly relevant for assessing threats and designing security mechanisms.

- A larger number of participants can be involved in individual transactions.
- Participants may not know who they are trusting.
- Participants may have less basis for trust.
- Because a grid is a large and complex distributed system, there are more individual points of attack.

Attackers often leverage access obtained on one insecure system to gain access to another system. While this is true in all systems, it is particularly relevant in grid systems because of the large number of participants.

Design goals for security in grid systems which are different than in traditional client–server systems include the following:

- Detection that an attack has occurred, and where in the system it occurred, is necessary in large grid systems, and it is a key part of an overall prevention strategy. Determining precisely where in the system the attack occurred is equally important but may be easier, and therefore more likely to be taken for granted, outside of the grid setting.
- It is important to detect problems in order to keep them from propagating through the system. It is important to be able to determine where the problems occurred in order to be able to prevent the problems from reoccurring.

- Additional attack prevention mechanisms are needed.
- There is a fundamental difference with respect to security between the interactive user of a host and the submitter of a batch job which runs on that host. Both need to be protected from the other, and this should be reflected in the system design.

Chapter 3

Background

This chapter presents background material describing the systems that the framework described in subsequent sections relies on and extends.

The framework is built upon the Condor high throughput computing system, developed at the University of Wisconsin–Madison and widely used in scientific and business computing. The first section describes Condor and how it works, including a history of the features of Condor related to security. Section 3.2 describes ClassAds, a language permitting expression evaluation used widely in Condor. Sections 3.4 and 3.5 describe the cryptography behind the security mechanisms introduced here. Section 3.6 introduces the Grid Security Infrastructure (GSI), an extension of X.509 PKI for job delegation in the Grid environment, which the framework extends and builds on.

3.1 Condor

Condor is a distributed batch computing system that has become widely used. Although it has been developed over a long period of time, it has recently been released in source form, with an Apache license, contributing to the popularity of the system.

Condor permits users to submit batch jobs to a centrally coordinated network of schedulers, which assume responsibility for the job, finding and scheduling resources, transferring files, and rescheduling as necessary when jobs do not complete successfully.

Condor installations are organized into groups of schedulers and execute machines called "pools." Advanced features permit inter-pool communication and job and resource sharing.

However, initially, the system was designed and implemented in the context of a university department, combining the resources and workload contributed by different research groups. Generally, each research group had a sequence of deadlines for conference and journal submissions, and a spike in computational demand in the weeks before each deadline. Because often these spikes did not completely overlap, the availability of other groups' resources permitted each group to reduce the amount of resources they would need to purchase and maintain.

3.1.1 Components

The central components of the Condor system, and the core of every Condor pool, are the `condor_collector` and `condor_negotiator` daemons. These components often run on the same server, although they can be run on separate servers, and can be made fault tolerant by running copies across several servers, through the use of a special high availability daemon `condor_had`. Together we call these central components, however they are configured, the *Central Manager* (CM) (the term is used even when multiple servers are employed).

In addition to the CM there are two other machine roles that are essential to the use of the system, the scheduler and execute roles. These roles are performed by the `condor_sched` and `condor_startd` respectively. The scheduler daemon accept jobs from users (and other schedulers when Condor-C is configured) and manage them on behalf of the user. The scheduler process is responsible for spooling input, intermediate, and output files and monitoring the job status as it executes. `condor_startd` process manages the direct execution of the tasks by providing a sandbox environment and spawning processes to manage execution. The `condor_starter` process is responsible for spawning, monitoring, and killing the tasks themselves.

3.1.2 History of Development

When it was first designed, Condor was used in a setting where machines, including workstations and servers, and user accounts were centrally administered by a single group [101].

As a result, security mechanisms were designed to suit this setting: users authenticated to an existing centralized system to obtain access to interactive machines and other resources such as filers and printers, so Condor used this existing authentication mechanism as an implicit authentication. For example, at the Computer Sciences department at the University of Wisconsin–Madison, Kerberos is used to authenticate interactive users; once users at this site are logged in to a workstation or server they can submit Condor jobs.

Later, explicit mechanisms were added to Condor; however this design persisted: the mechanisms within the distributed system responsible for managing jobs and resources, finding matches and starting jobs, are all presumed to be trusted by the system. This assumption, which was appropriate for the environment in which early versions of Condor were developed, is no longer appropriate when Condor is used in environments consisting of multiple administrative domains.

The Condor system has increasingly incorporated features that make it able to work in more complex environments which span administrative domains [101]. For example, *flocking* and *Condor-C* are mechanisms for moving jobs from one administrative domain to another. With flocking, schedulers from one administrative domain that have more queued jobs than available resources advertize those jobs to the central manager of another domain. Condor-C is a mechanism for moving jobs from one scheduler to another. In fact, because of its flexibility and reliability, Condor is often used to connect other systems, such as commercial batch queueing systems and Globus Toolkit. However, although these mechanisms exist, the underlying assumption that all parties trust the administrators persists.

3.2 ClassAds

The *ClassAd* language is a simple language used within Condor and other systems to store metadata for tasks and resources. ClassAds are sets of name-value pairs; the values can contain expressions that can refer to names within the ClassAd, as well as names in other ClassAds. The ClassAd language defines the structure of ClassAds and the rules for expression evaluation. An example ClassAd is shown in Figure 3.1.

ClassAds are the *lingua franca* of the Condor system: every resource and task has an associated ClassAd which describes it in detail. At present, ClassAds are not explicitly protected; in the proposed system, ClassAds can be digitally signed. Digital signatures can be used to detect and deter tampering. For example, when a host receives a task to be executed, it can verify for itself who submitted the task and that it is unaltered.

```
Cmd = /full/path/to/executable;
TransferInputFiles = this_file.txt,that...;
Requirements = (Arch == "INTEL") && ...
```

Figure 3.1 **An example ClassAd.** Note that the `Requirements` attribute specifies an expression.

3.3 CEDAR

Point-to-point authentication, session key establishment, and session integrity and confidentiality are handled in Condor by CEDAR, a library that permits two communicating processes to negotiate security settings such as authentication methods and cryptographic algorithms. CEDAR is similar to SASL [68].

Integrity is provided via MD5 [82]; support for stronger checksum algorithms is planned but not currently implemented. The following encryption algorithms are implemented:

- Blowfish [89]
- Triple-DES [1]
- AES-128, AES-192, AES-256 (implemented but not yet released in Condor) [30]

The following authentication methods are implemented:

- Claim-to-be - a stand-in method for testing connectivity. Not intended for use in production environments.

- Kerberos [96]
- SSL [33] - see below.
- GSI [39] - see below.
- Password - communicating parties must have established a shared secret before authenticating. The implementation of this method is based on the AKEP2 protocol [67, 22].
- Microsoft's SSPI [2]

3.4 Cryptographic Primitives

The work presented in this thesis draws on the existing cryptographic primitives including cryptographic hash functions, symmetric encryption, and public key cryptography. This section describes the usage of these primitives, introduces notation, and describes basic functions (such as digital signatures and certificates) based on these primitives. The notation is summarized in Table 3.1.

For each function, the complexity of the function in terms of the size of the input messages is included as well. I assume that the key size and the algorithms used are fixed. Section 3.6 describes the algorithms and key sizes used in Condor and GSI.

Using similar notation, the sections of Chapter 4 describe the functions introduced in this thesis.

3.4.1 Cryptographic Hash Function

A cryptographic hash function is a function

$$H = \text{hash}(M),$$

that takes a sequence of bits M, and produces a fixed-size bit string H, and has the following properties:

- Given H, it is not feasible to find Mor any substring of M.

Symbol	Meaning
H	A cryptographic hash
M	A message or string of bits
E	An encrypted message or string of bits
S	A secret key
P	The public portion of a keypair
K	The private portion of a keypair
B	A fixed size block of text
D	A digital signature

Table 3.1 **Notation for cryptographic primitives.**

- Given M, it is not feasible to find a collision (i.e. a different string M' that hashes to the same string H).
- The hash can be computed in $\mathbf{O}(|M|)$.

3.4.2 Symmetric Encryption

A *symmetric encryption* algorithm provides a pair of functions, *encrypt* and *decrypt*:

$$E = \text{encrypt}(M, S)$$

and

$$M = \text{decrypt}(E, S)$$

The encryption function takes as input two strings, one an arbitrary length message M, and the other a fixed size key S, and produces an encrypted message E, approximately the same size as M. A symmetric encryption algorithm has the following properties:

- Given E, it is not feasible to find Mwithout knowing S.
- The message can be encrypted and decrypted in $\mathbf{O}(|M|)$.

3.4.3 Public Key Encryption

A *public key encryption* algorithm also provides functions encrypt and decrypt, but instead of using a symmetric key S, they take a keypair $\langle P, K \rangle$, where the private portion of the key is known only the holder but the public portion can be published widely:

$$E = \text{encrypt}(B, P)$$

and

$$B = \text{decrypt}(E, K)$$

Because the public key encryption and decryption functions are so much more expensive to calculate than the symmetric equivalents, these functions are usually performed on small messages (represented here as B). When larger messages need to be encrypted using public key encryption, they are usually encrypted using a symmetric key which is then encrypted using the public key.

$$E = \langle \text{encrypt}_{\text{pub}}(S, P), \text{encrypt}_{\text{sym}}(M, S) \rangle$$

and

$$S = \text{decrypt}_{\text{pub}}(E_1, K)$$

$$M = \text{decrypt}_{\text{sym}}(E_2, S)$$

Public key encryption algorithms are also used for *digital signatures*, by reversing the usage of the keys: the private portion of the keypair is used to encrypt a block; anyone in possession of the public portion, along with the block and the encrypted block, can verify that the result was encrypted by the holder of the private portion by decrypting the encrypting block and directly comparing with the unencrypted block:

$$D = \text{encrypt}(B, K)$$

and

$$B = \text{decrypt}(D, P)$$

By combining these functions with hashing, the following sign and verify functions are obtained:

$$D = \text{sign}(M, K) = \text{encrypt}(\text{hash}(M), K)$$

and the boolean

$$\text{verify}(\langle M, D \rangle, P) = \text{hash}(M) \equiv \text{decrypt}(D, P)$$

3.5 Public Key Infrastructure

A *Public Key Infrastructure* (PKI) is a system for identifying users and processes based on public key cryptography. X.509 is a widely used standard for implementing a PKI. An X.509 PKI is based around a set of hierarchies of *certificates* each of which identifies a principal, or end-entity. Support for X.509 PKIs is implemented in Condor in the form of support for GSI and SSL authentication methods.

3.5.1 Certificates

An X.509 certificate is a *binding* between a keypair and a name, permitting the principal holding the private key portion of the keypair to identify themselves to others. The binding itself is in the form of a signature by a third party explicitly trusted by the recipient of the certificate to authoritatively identify the key holder.

A certificate is a digitally signed message with a specific structure; the signed message includes the subject name and their public key, along with fields identifying the signer and indicating the validity period of the certificate. The X.509v3 standard describes a set of required fields and includes an extension mechanism that permits additional fields to be included. An issuer of certificates is called a *Certificate Authority* (CA).

For example, a certificate linking the name "A. User" with a particular keypair, issued by a CA with the name "B. CA" would be:

$$C_{\text{A. User,B. CA}} = \text{sign}(\langle \text{Sub:A. User}, \text{Sub Key:} P_{\text{A. User}}, \text{Issuer:B. CA}, \ldots \rangle, K_{\text{B. CA}})$$

One field within a certificate indicates whether the certificate is permitted to be used to issue other certificates: the *CA bit*. If the CA bit is set, then the certificate identifies a CA. When a certificate is signed by the private key that forms a keypair with the public key contained in the certificate, the certificate is said to be *self-signed*. If a certificate does not have the CA bit set, it is called an *end-entity certificate*.

To verify a certificate, the issuer must also be verified and trusted. The signature on the certificate itself is checked using the public key from the certificate of the issuer. The issuer's certificate may be a self-signed CA root certificate explicitly trusted by the verifier, or an intermediate certificate. A *certificate chain* consists of a sequence of one or more certificates each signed by the next. In other words, for a certificate to be valid, it must begin a chain of one or more certificates for which the following conditions are met:

1. The signature on the certificate must be valid.
2. The signature on the certificate must have been made by the issuer.
3. The issuer certificate must be a valid certificate.
4. The last certificate in the chain must be a CA root certificate explicitly trusted by the verifier.

Since CAs themselves are identified by certificates, trees of hierarchies are formed, with CA root certificates at the root, end-entity certificates as the leaves, and intermediate CA certificates in the middle.

The protocol for issuing certificates does not require that the certificate holder expose the private part of the certified keypair to the CA. Instead the holder prepares a *certificate signing request* (CSR) which contains the public key, the proposed subject name, and is signed with the private key. It is the CAs responsibility to authenticate that the entity presenting the CSR is in fact the entity identified by the subject name in the CSR. The mechanisms for performing this authentication are not specified and vary widely in practice, depending on the requirements of the PKI.

3.6 The Grid Security Infrastructure

Note that within an X.509 PKI, end-entity certificates may not be used to sign other certificates.

The Grid Security Infrastructure (GSI) extends the PKI concept to include *proxy credentials*, intended to allow tasks in Grid environments to authenticate on behalf of the users who have submitted the tasks [39]. This is accomplished by relaxing the restriction on end-entity certificates so that they may issue proxy certificates, in effect acting as CAs. In addition, proxy certificates may also issue further proxy certificates, except when they are explicitly restricted from doing so.

Proxy certificates are intended for use by processes acting on behalf of a particular user, so that the process (i.e. jobs submitted to a batch computing system) can authenticate with the credentials of the user without risking exposure of the user's X.509 private key. In practice, proxy certificates are limited in validity time in order to make the increased possibility of exposure of the proxy private keys an acceptable risk.

The process of issuing proxy certificates is much the same as the process of issuing regular X.509 certificates: the recipient is not required to expose their private key to the issuing process (whether that process holds an end-entity certificate or a proxy certificate).

3.7 The Task Pathway

In this section we describe a simplified task pathway, shown in Figure 3.2, including the components of the Condor system that handle a task as it moves through the system and the interactions between these components. We describe the security mechanisms in place (without the contributions in this thesis) and the trust assumptions that are made.

3.7.1 Roles

The Condor system consists of several processes distributed across a network of machines; each process plays a specific role within the infrastructure. For simplicity, we consider a

limited set of roles, where each role may be performed by a set of interacting processes. The basic roles are:

- The *submitter*. This is the process the user uses to submit tasks. At this stage in the task workflow, the executable programs, arguments, and input files are specified.
- The *scheduler*. This role includes a process for matchmaking, used to match tasks to resources, a process for collecting information about tasks and resources, and a process for receiving tasks and distributing them to resources. Within Condor, when a task crosses administrative boundaries, it moves from a scheduler in one domain to a scheduler in another.
- The *worker node (WN)*. This is where tasks are performed.
- The *storage element (SE)*. Often worker nodes access storage elements to obtain input data or store output.

3.7.2 Actions

[1]

We assume that point-to-point communications occur over a secure channel provided by CEDAR, a mechanism in Condor that provides mutual authentication, integrity, and confidentiality. Although Condor supports several authentication methods, we assume that a PKI has been established: that the GSI authentication method [39] is used between all communicating processes, and that a certificate authority (CA) has been used to generate certificates for each process and user.

There are three types of actions that can occur on behalf of or to transfer a task:

- A *forwarding* step occurs when a task moves between a submitter and a scheduler, between schedulers, or between a scheduler and a WN.
- An *execute* step occurs when a WN executes a task.

[1] TODO: This needs to be changed to reflect the transparent credential transformation section.

- An *access* step occurs when a worker node accesses a storage element to read or write data.

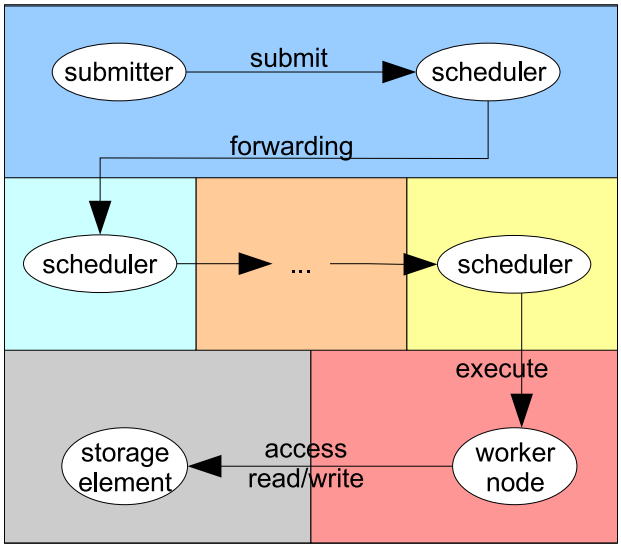

Figure 3.2 **The task pathway.** Regions represent potentially different administrative domains. One or more schedulers may handle the task between submission and execution.

3.7.3 Checks

When a request for service from one process to another is made, several checks are made regarding the permissibility of the request. In particular, the mutual authentication performed between communicating parties includes some implicit authorization: if communicating parties are unable to authenticate each other, the request is dropped.

In Condor, when a process authenticates with another process, each identifies the other using a canonical identifying string of the form `user@host.domain`. Per-process coarse-grained authorization policies specify a set of identities that can read or write (i.e., make requests that read or write data) from the process.

- *submit*: The submitter explicitly selects the scheduler to which they submit their job. The scheduler is authenticated using the certificate chain trusted root certificates

configured by the user and can be additionally limited by the `GSI_DAEMON_NAME` configuration variable, which specifies a list of acceptable daemon DNs. The user implicitly trusts the scheduler it contacts and any schedulers and WNs that it may choose to forward the job to.

- *forwarding* from scheduler to scheduler: The submitter may explicitly specify a destination scheduler, or the administrator determines a set of potential recipient schedulers; selection is performed by the matchmaking process. The scheduler who forwards the job authenticates with the delegated credentials of the submitting user while the recipient authenticates with its service credentials; another delegation step is performed. When the *execute* step below completes, the recipient scheduler returns the data produced in that step to the originating scheduler.

- *forwarding* from scheduler to WN: The scheduler selects the worker node as part of the matchmaking process, which takes into account requirements specified by both the worker node and the submitting user and finds matches between the two. In order to facilitate access control based on the scheduler (in addition to the user) at this point the scheduler authenticates to the WN with its service credentials rather than with the credentials of the user. A delegation step is performed with the proxy credentials of the user so the WN can impersonate the user. When the task is complete, the results are returned to the scheduler.

- *execute*: The WN performs this step internally; no additional delegation step is performed. At this point, the WN checks the authorization of the user before executing the task.

- *access*: File reads and writes are explicitly under the control of the task, and therefore assumed to be under the control of the submitter. The worker node authenticates to storage elements using the delegated proxy credentials of the submitting user.

3.7.4 Analysis

The security protections described in this section inadequately protect the three sets of data described in the introduction: the task input and output data, the operating environment of the worker node, and data unrelated to the task, such as that stored on an SE.

We assume that the submitters (i.e. all of the processes directly controlled by the submitter) are trustworthy; in addition we assume that each submitter is able to specify a set of worker nodes and additional resources such as storage elements they trust.

In our threat model, attackers can control processes within the infrastructure and will use this control to perform attacks on other processes in the infrastructure and to attempt to obtain unauthorized access to data. We are primarily concerned with integrity and confidentiality, since attacks on availability are relatively easy to detect in the Grid environment.

Despite the security protections in place in the task pathway, the design of the existing security mechanisms violates the principle of least privilege [87]: "Every program and every user of the system should operate using the least set of privileges necessary to complete the job." The system design assumes that *each user considers each process in the system to be completely trustworthy*: that each process will protect task availability, integrity, and confidentiality, and that the user's credentials will not be exposed or misused to obtain access beyond the usage required for the task submitted by the user.

For example, if a scheduler is not trustworthy, it can read and copy task data or results, or alter the contents of the task. The altered task could be used to attack the worker node, or to read or write any data that the user's delegated credential is authorized to access on storage elements. The only limit on the usage of the delegated credentials is the lifetime of the proxy. Even if the task makes it unaltered to a worker node, the results can be altered while they return to the submitter.

Chapter 4

Framework Components

In this section, each of the mechanisms used to enforce policy within the framework are described in detail. In the next section, the protocols used to integrate these mechanisms are discussed.

4.1 Overview

The framework consists of a set of mechanisms which work together to enforce policies. These mechanisms include:

- The application of digital signatures to task data and metadata.
- The use of proxy certificates that are cryptographically bound to individual tasks and services.
- A policy language with clear evaluation rules that allows users to specify how their tasks and credentials may be used.
- A service for producing the type of authentication tokens on which our system relies (GSI proxy certificates) for installations which use alternative authentication mechanisms such as Kerberos, SSL, shared passwords, or SSPI / NTLM.
- Techniques for providing confidentiality for tasks while in transit.

In order to facilitate adoption, these mechanisms work by adding information to existing task metadata and by extending proxy certificates. As a result, existing systems which

are not aware of the mechanisms continue to work as they do now, permitting incremental deployment. These mechanisms have been implemented in the Condor system [100].

Although the framework is described in the context of Condor, it could readily be implemented and deployed a more general Grid environment. The chapter covering implementation describes the steps required to achieve this in greater detail; see Chapter 6. Condor (and related components, such as ClassAds) are described in more detail in Section 3.

A key mechanism on which several other mechanisms rely on is per-task credentials. In existing systems, credentials are issued to individual users and to daemons or hosts. By permitting users to link delegated credentials to a specific task, this technique allows users to specify policies associated with the individual tasks they submit, and restricts the ability to misuse user credentials for other tasks.

The result of linking credentials directly to tasks is a reduced need for the endpoints to rely on the central infrastructure. In existing systems, the central infrastructure is trusted to provide integrity for task data (both results and input data, including the task executable itself) and to maintain the association between tasks and authenticated users.

Another mechanism we introduce is a certificate which is part of one delegation chain but also includes information linking it to another certificate. This service-specific proxy certificate is used to identify both the user who submitted the delegated task as well as the host on which their task is running. As a part of the framework, service-specific proxy certificates allow a process running on a particular host on behalf of a particular user to authenticate as such when communicating with a file server or other resource that is configured to enforce the user's policy regarding where their jobs can be run.

Drawing on the end-to-end principle [86] and the principle of least privilege [87], the framework is intended to reduce the amount of trust that participants (job submitters and resource owners) must be willing to grant to other participants.

The framework is intended to permit users to safely submit tasks to a distributed system in which they trust only a subset of resource owners by providing users with the ability to specify where their jobs can run and to verify that their policies have been followed. The

ultimate goals of this work are to both help to secure existing deployed systems and to permit sharing in contexts where sharing is not currently possible.

The framework consists of the following independent mechanisms:

- *Signed ClassAds*
- *Task-specific proxy credentials*
- *Action authorization expressions*
- *Service-specific proxy credentials*
- *Transparent credential transformation*
- *Secure key storage service*

4.2 Signed ClassAds

A ClassAd is a set of attributes and associated values. The ClassAd language defines a syntax for ClassAds and rules for expression evaluation. ClassAds are used within Condor to specify attributes associated with tasks, resources and processes. ClassAds are described in more detail in Section 3.2.

Signed ClassAds extend ClassAds by adding support for applying and verifying digital signatures using X.509 private keys and certificates. Signed ClassAds provide integrity verification capability within the framework. In addition, signed ClassAds can play a role in authentication and authorization, and can assist audit capability.

Signed ClassAds are implemented by applying basic digital signature techniques to string representations of ClassAds. See Section 3.4 for a detailed description of digital signatures and cryptography. The signature value is a ClassAd attribute with the name *SignedClassAd* and the value containing the entire text to be signed, followed by the cryptographic checksum of that text encrypted with the private key of the signer. See Chapter 6 for more details regarding the specifics of the cryprograpic techinques employed, the format of the

SignedClassAd attribute, and the methods provided for creating signatures, validating the results, and obtaining the original signed data.

Conceptually, a signed ClassAd is simply a ClassAd which has a subset of attributes that have been digitally signed using a private key. This key is associated with an X.509 or GSI end-entity certificate or a GSI proxy certificate. (X.509 and GSI credentials are described in more detail in Sections 3.5 and 3.6, respectively), and the meaning of the signature is that the entity named in the certificate (the holder of the private key) is linked to the contents of the ClassAd: that the contents of the ClassAd are accurate according to the signer at the time of signing.

In practice, the role of the signer dictates the meaning of the signature; for example, a signature on a job ClassAd links the signer with the specific task, while a signature on a machine ClassAd links the specifics of a particular resource offering (such as available memory) with a particular offering party (the entity controlling the machine). However, in the same way that the ClassAd language describes a format for sets of attribute-value pairs, an expression language and semantic rules for evaluating those expressions in the context of pairs of ClassAds, these meanings are not inherent in the mechanism, they are inherent in the use of the mechanism within the framework. See Chapter 5 for more detail on how this mechanism is used in Condor and integrated with the framework we introduce.

A recipient of a signed ClassAd in possession of nothing more than a Certificate Authority (CA) root certificate should be able to verify whether or not the ClassAd was signed by a credential issued by, or delegated from, that root certificate, and whether or not the ClassAd had been altered. A recipient without a CA root certificate should be able to tell whether the ClassAd has been altered.

File references (in the form of a filename and a cryptographic hash of the file contents) can also be included within the data that is signed, so external files such as executables and input files can be included in the signature.

An example signed ClassAd is shown in Figure 4.1.

```
Cmd = /full/path/to/executable
Cmd_Sha1Sum = "97a3e984d6..."
TransferInputFiles = this_file.txt,that...
TransferInputFiles_Sha1Sum = "8a552f...",...
Requirements = (Arch == "INTEL") && ...
Args = -rf
SignedClassAd = "[Cmd ...-rf],b70a6..."
```

Figure 4.1 **An example signed ClassAd.**

Signed ClassAds can play a role in authentication, authorization, and audit mechanisms: they make it possible to establish a cryptographic link between the authentication information accompanying the ClassAd and the actual contents of the ClassAd as received. Authorization should fail if relevant contents have been altered. Because authorization information specified by the signer can be included in the signed ClassAd (see Section 4.4), policy enforcement points can use signed ClassAds to make authorization decisions on behalf of the signer. Audit mechanisms are strengthened by explicit association between a task and information about its origin, and between resources and the tasks they perform.

4.3 Task Specific Proxy Certificates

A task-specific proxy credential is a key pair consisting of a private key and a GSI proxy certificate with a signed job ClassAd embedded into it, linking the credential and the job. The proxy certificate includes the signed job ClassAd within the policy section of the certificate. The X.509 proxy certificate standard includes support for a section containing policy; this section includes a policy identifier and a variable length string of bytes containing the policy itself [103]. The standard requires that an entity authenticating a proxy containing a policy must be able to interpret the policy and that it should refuse to accept the proxy if the policy language can not be interpreted and enforced by subsequent authorization steps.

The policy language is specified as OID; we have registered an OID arc for the Condor project (1.3.6.1.4.1.214.102) and have designated an OID specifically for task-specific proxy certificates (1.3.6.1.4.1.214.102.1), and service-specific proxy certificates (1.3.6.1.4.1.214.102.2), described below.

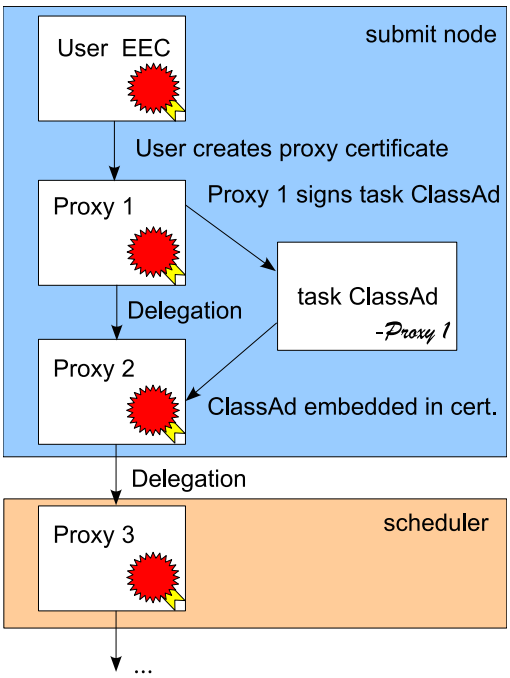

Figure 4.2 **The task-specific proxy certificate.**

Before submitting a task, the submitter creates a proxy credential specifically associated with that task by embedding a signed job ClassAd within the proxy certificate, creating a *task specific proxy certificate*. The steps in creating a task-specific proxy certificate are shown in Figure 4.2. First, the user creates a proxy certificate from their end-entity certificate (EEC) as they normally would. Then, for each job they submit, they first create a signed ClassAd, containing any policy restrictions (see Section 4.4, below) and the names and checksums of any files such as executables or input files. This ClassAd is signed by the proxy certificate. Then, they perform a delegation step, creating the keypair locally, but including the ClassAd within the policy field within the delegated proxy certificate.

While, in theory, it would be possible to create a task-specific proxy certificate directly from the EEC, this would require a user to type their password for every distinct task they submit. A task specific proxy certificate must be issued by the first proxy in the chain (and the signed ClassAd it contains must also be signed by that proxy).

Task-specific proxy certificates were introduced in [7].

Although we doubly sign the ClassAd (once as described above, in Section 4.2, and once by embedding it in a certificate), this is merely because these signatures are used by different

components at different times; if analysis showed that the overhead is too great, we could merge these signatures. As described in Section 5.5, the presence of the text of the ClassAd in the credential chain permits the policies there to be used to perform authorization without transferring additional data beyond the certificate chain already used for authentication.

Task specific proxy credential chains (credential chains which contains task specific proxy certificates) are intended for use by WNs to verify the integrity of the associated task before execution, or for use by the task to access external resources. However, a scheduler can safely ignore the embedded policy; in this sense the certificate expands rather than restricts what the certificate can be used for. Enforcement is provided by the trusted components aware of the protection mechanisms within the system including trusted worker nodes, the submitter, and trusted storage elements. See Section 5 for details about how enforcement is performed.

4.4 Action Authorization Expressions

An action authorization expression is an expression in the ClassAd language that expresses policy determining whether a particular action should be performed. In combination with task-specific proxy credentials, these expressions permit users who submit tasks to specify policy about the disposition of their jobs.

The submitting user can specify conditions on the *execute* and *access* actions described above in Section 3.7.2 The conditions are specified as expressions in the ClassAd language and are enforced by the trusted components: the submitter, the WN, and the SE. The expressions limit which actions can be performed on which objects by which principals. These policy expressions are Boolean expressions that evaluate to true or false: if (given the action and environment) a expression evaluates to true, the action can proceed.

To support these expressions, information about the identities of the participants (and in the case of the *access* action, details about the objects being accessed) will be added to the ClassAd by the enforcing process before the expression is evaluated. For example, the

distinguished names of the WN and the issuer of the WN's certificate are available through attributes.

The *execute* action expression is used to specify the set of trusted certificate holders that the user accepts as trustworthy to perform the computation correctly when given a valid task, to preserve the integrity of the task input data, executable, and output data, and to preserve the confidentiality and integrity of data manipulated through resources such as SEs on behalf of the submitting user. A trusted WN is defined as a WN for which this expression evaluates to true, but we do not rely solely on WNs to enforce this policy: the policy is enforced at the submitter and on any resources the task attempts to access; see Section 5 for details. For example, a simple expression specifying acceptable execution hosts is shown in Figure 4.3.

```
regexp("/O=Fermi...", AAE_execute_issuer_DN) || regexp("/O=UW...", AAE_execute_DN)
```

Figure 4.3 **An execute action expression.**

The *access* action expression specifies a set of acceptable data accesses. The attributes supplied at enforcement time include paths to the objects being accessed and the access type (read or write). Complex expressions are supported given the expressiveness of the ClassAd language; often simple expressions are adequate. The signed access policy contained in the proxy chain acts as a distributed capability which is interpreted in conjunction with any access control lists stored within the resource to perform authorization and access control decisions.

4.5 Service-Specific Proxy Certificates

A service-specific proxy credential is a key pair consisting of a private key and a GSI proxy certificate with a signed machine ClassAd embedded into it. This credential is part of a chain can be used to identify the user submitting the task, the task itself, and the service associated with the machine ClassAd.

A *service-specific proxy certificate* is similar to a task specific proxy certificate described above in Section 4.3, but instead of identifying a particular task associated with the credential, it includes information about a key held by the service which controls the proxy certificate. When a task running on a trusted WN authenticates to a resource to perform an access, the resource often wants to be able to identify both the submitting user as well as the WN on which the task is running. However, authentication is performed using only one credential, and the task does not have access to the private key used to authenticate the WN. In order to identify the WN in a cryptographically strong way, we get the WN to use its private key to sign a piece of information which uniquely identifies the proxy certificate, and then include this signature in the proxy certificate.

When a WN receives a proxy delegation from a scheduler, the WN first creates a keypair to be used as the next credential in the proxy chain. The WN then creates an additional attribute that it inserts into the certificate signing request it gives to the scheduler, consisting of the public key of the keypair it just created, signed by the service or host key of the WN. By including a signature on the public key part of the keypair in the newly created proxy certificate, the WN claims control over the associated private key. In this way, the service-specific proxy credential is delegated by both the user and the host on which the credential is generated.

In order for this credential to meaningfully identify both the user and the WN, neither must expose the private key part of the delegated credential. The WN ensures that this will not occur by protecting it when it has control of it, and by making sure that the delegated credential is used only by processes explicitly signed by the submitting user: it is up to the submitting user to ensure that their task does not compromise the confidentiality of their delegated private key (since this mechanism protects their task and data).

4.6 Transparent Credential Transformation

Transparent credential transformation permits the other mechanisms which rely on PKI to be used for users who authenticate using other systems, by issuing PKI credentials at user request or without user involvement, depending on configuration.

One of the most flexible mechanisms within the Condor system is its authentication system. Users can authenticate themselves when they submit a job using GSI (X.509 certificates), Kerberos, Windows NTSSPI, SSL and simple passwords. Credential transformation is a mechanism for allowing the use of mechanisms which rely on GSI authentication and delegated proxy certificates to be used when users authenticate using other authentication types which do not directly support delegation, such as filesystem, password, SSL, NTSSPI or Kerberos (delegation using Kerberos is possible but not supported in Condor). The basic mechanism is simple: users authenticate to a daemon that issues them a GSI credential, which is then used in place of their existing credential when they submit a job or communicate with other components of the Condor infrastructure.

In order to digitally sign something, the signer must have a private key, and the verifier must have the associated public key. In order to provide this functionality while continuing to support a broad set of authentication mechanisms, we have implemented a certificate requesting client and online CA (certificate authority) service. Users can request a PKI certificate after authenticating using another of the supported authentication mechanisms.

In essence, the certificate produced by an online CA certifies that the holder authenticated at a particular time using a particular method. This allows a user who has authenticated using Kerberos, for example, which does not use keypairs and would be difficult to use for digital signatures, to sign their jobs. In addition, the certificate provided by the online CA attests their username and identity provider, and therefore a service such as `condor_sched` which trusts the CA can retrieve attributes about the user with confidence that the attributes they're retrieving actually apply to the submitter of the job they have.

4.7 Confidential Data and the Secure Key Storage Service

Protecting the confidentiality of a task in a Grid computing environment is a difficult problem, because worker nodes must have access to task contents in order to execute the task. One way to achieve confidence that a worker node is trustworthy is to require that it use Trusted Computing [64, 63]. We assume that each user has confidence that some set of worker nodes are trustworthy; our focus is on protecting confidentiality from other infrastructure components.

Submitters can include an encrypted payload with the tasks they submit. The payload is encrypted with a randomly generated symmetric key that is not included with the task, but rather stored with an agent of the submitter, the `condor_credd` process. When a worker node receives an encrypted payload, it contacts the agent directly and authenticates using its SSPC. The key is returned via an encrypted CEDAR session to the worker node.

Because the `condor_credd` authenticates using the delegation chain terminating in the SSPC and the TSPC, it is able to evaluate the AAEs contained in the chain to verify that releasing the key is appropriate and that the host on which the job is running is acceptable to the submitter.

Encrypted payloads provide the confidentiality component of the framework. If the cost of encryption outweighs the need for confidentiality, encrypted payloads need not be used. In addition, depending on file-staging requirements, confidentiality can be achieved by transferring sensitive files directly between storage elements and worker nodes over an encrypted connection. Although we have not implemented this in the context of this thesis, it would be possible to outfit a storage element with the framework SSPC/AAE verification mechanism just as we have done with the `condor_credd`.

The confidentiality of encrypted payloads and direct transfer depends on the ability of the framework to help the submitter, the credential agent, and the storage element distinguish between trusted worker nodes and other infrastructure processes.

To permit maximum flexibility, we do not explicitly encrypt payloads directly in the code we implement; rather, we implement mechanisms that can be used to distribute the shared secrets used to encrypt the payloads. In addition to reducing the complexity of the implementation, this permits the users to choose the implementation of the encryption mechanisms to best suit their needs.

There are a number of reasons why this may be more desirable. A user may choose to encrypt more sensitive portions of their task with more expensive encryption algorithms and less sensitive portions with less expensive algorithms or none at all. A user who is submitting a set of related tasks may choose to encrypt a portion of each task with a task-specific secret while encrypting the rest of each task with a secret common to the set.

This mirrors the flexibility permitted in the built-in point-to-point encryption mechanisms included in Condor, and permits a variety of performance/security choices for end-to-end confidentiality

Chapter 5

Framework Integration

This chapter describes the changes to the task pathway and how the framework mechanisms described in Chapter 4 are integrated into the task pathway and with each other. In order to illustrate the integration, an example showing the use of the confidentiality mechanism is given. The confidentiality mechanism is illustrative because the it takes advantage of most of the other mechanisms, including signed ClassAds, task-specific proxy certificates, service-specific proxy certificates and action authorization expressions. For completeness, we also show how transparent credential transformation would be integrated into this workflow.

In the example, the following security policy is implemented: a job may only be run on a set of machines whose host certificates were issued by a particular Certificate Authority (CA). The intermediaries may be untrusted so the job should be encrypted before submission, and hosts which execute the task should verify the integrity of the jobs they receive before they execute them. The encryption key must be provided to the jobs so that they can run, but should only be provided to jobs that can demonstrate that they are running on a host which is in possession of a credential issued by this CA. When the job is complete, the submitter will reject it if the result is not signed by a host identified by this credential. The submitting user is logged in to the system they will submit from using Kerberos credentials and does not have GSI credentials.

In other words, the user trusts the key infrastructure, the submit machine, a set of execute machines, and the `condor_credd` host, and not the other components of the system; attacks on availability (e.g. denial of service) are not considered.

The remainder of this chapter shows how each of the framework components contribute to the implementation of this policy. Chapter 7 provides detailed performance analysis of the example described here.

At a high level, the user uses the MiniCA (transparent credential transformation) to obtain GSI credentials, invokes `grid-proxy-init` to obtain a base user-specific proxy certificate, encrypts their job as necessary, and invokes `condor_submit` configured to provide signed job ClassAds and task-specific proxy credentials and store the secret used to encrypt the job for confidentiality. The `condor_starter` process on the execute host verifies the task-specific credential and the signature on the job (and that the two match), and issues a service-specific proxy credential which it provides to the job wrapper. The job wrapper uses the SSPC to authenticate to the `condor_credd` to obtain the secret, decrypts the job and runs it. The job wrapper then encrypts the results before exiting. When the job is complete, `condor_starter` signs the results of the job. The user retrieves the results and checks the signature and the identity of the host credential signing the job.

5.1 Transparent Credential Transformation

In order to demonstrate all of the components of the framework, we assume that the user does not have a GSI credential (an EEC) and will use a transparent credential transformation daemon to obtain one. The transparent credential daemon process is known as the MiniCA and is `condor_minica`. Although other components of the framework require credentials such as those issued by the MiniCA, there are other ways to obtain those credentials. (See Section 3.6 for more information about GSI.) This step is essentially independent of the other steps and occurs only once when new GSI credentials are needed. Any of a number of authentication methods could be used; we illustrate with the Kerberos authentication method, already included in Condor.

For brevity, we do not describe here the complete process for setting up a MiniCA but the setup is simple: a script creates a self-signed root key, directory structure and configuration files. The CA is based on the simple CA included with OpenSSL. It is important to secure

the host on which the MiniCA daemon runs appropriately, since the confidentiality of the CA root key is an essential requirement.

While `condor_minica` uses standard Condor authentication methods, it uses a special authorization mechanism permitting administrators to configure which users can obtain credentials based on how they authenticate and who they are.

The user runs the following command to obtain credentials from the MiniCA:

```
condor_minica_client -C alderman@cs.wisc.edu -f minica/alderman
```

The client uses the `condor_config` file to determine the location of a custom OpenSSL configuration file prepared by the administrator. The client first creates a keypair with a private key and a certificate signing request. The certificate signing request for a user certificate is constructed with the Common Name portion of the DN set to the user's login name and the `UID_DOMAIN` from the `condor_config` file, in the form `uid@domain.edu`. Then it contacts the `condor_collector` to determine the address of the `condor_minica` daemon, then connects to that daemon. The client authenticates with the daemon using Kerberos (any authentication method supported by Condor can be used here). The result of the exchange with the MiniCA is a keypair consisting of a private key and an end entity certificate (EEC) signed by the MiniCA. In our example, the DN takes the form:

```
C=US, ST=WI, O=Condor, OU=Research, CN=alderman@cs.wisc.edu
```

5.2 At the Submitter

The user prepares to submit the task by creating a proxy certificate. The first proxy certificate that is created is a normal (i.e. not task specific) proxy certificate, and may be used to submit more than one task. This is produced using the GSI tool `grid-proxy-init`.

The following steps are performed at submit time in this order:

1. The user creates a secret key for encrypting the job.

2. The user encrypts the job. A simple wrapper script is generated that (when run on an appropriate WN) accesses the `condor_credd` process and decrypts and execs the real job. This simple script replaces the real job.

3. The user invokes `condor_submit`. This tool performs the following steps:

   (a) Create and sign the job d with the user's proxy credential.

   (b) Create the task-specific proxy credential from the user's proxy, and the signed ClassAd.

   (c) Contact the `condor_credd` to register the secret key used to encrypt the job.

   (d) The task-specific proxy credential is used to actually submit the job.

The configuration file used by the submitter contains the following entries:
Controlling signing:

```
SIGN_CLASSADS = True

SIGN_JOB_CLASSAD_ATTRIBUTES = "Owner, CondorVersion, CondorPlatform, \
Iwd, JobUniverse, CmdHash, CmdHashType, Arguments, ExecutionHostAAE"

CLASSAD_SIGNATURE_CREDENTIAL_TYPE = GSI
```

Controlling hashing:

```
CMD_HASH = True

CMD_HASH_TYPE = sha1
```

To generate the TSPC:

```
ADD_TASK_POLICY = True
```

In the submit file (usually managed by the submitting user), or in the configuration file used by `condor_submit` (usually managed by an administrator), action authorization

expressions are included defining the policies for running the job. As described in the introduction to this chapter, for this example, these expressions are intended to limit the set of machines that can execute this task. Specifically, we want to limit the set of machines to those possessing host credentials with a DN that matches the regular expression:

`^C=US/ST=WI/O=Condor/OU=Research/CN=glow-s\d\d\d\.cs\.wisc\.edu$`

When the ClassAd for the task is generated, the user specifies action authorization expressions for the *execute*, and *access* actions. (These policies can default to values set by the site administrator.) To implement the limit described above, the following line would be added to the submit file:

`+ExecutionHostAAE = regexp("`*`regex above`*`", ExecutionHost)`

When this expression is validated, the `ExecutionHost` attribute will be instantiated by the verifier, and will be undefined if the host certificate used to sign the policy field of the SSPC descends from one of the trust roots configured at the verifier. For more on SSPC verification see Section 5.5.

Techniques may be developed to automatically generate these expressions from a more general policy language and to automatically update other relevant ClassAd settings, such as the requirements expression. The action authorization expressions are included in the task ClassAd along with references to, and checksums for, any external files that are to accompany the task, including input data and the task executable.

When submitter process contacts the scheduler to submit the task, mutual authentication is performed; the submitter authenticates using the task-specific proxy certificate and the scheduler authenticates using its process credentials. If authentication and authorization (as described in Section 3.7.3) is successful, delegation proceeds.

5.3 At a Scheduler

Once the task is received by the scheduler, the regular matchmaking process can begin and proceed as normal. When a match is found, the authentication, authorization, and delegation sequence repeats. The scheduler ignores the policy contained in the TSPC.

In our research implementation, the user has the responsibility of making sure that the matching, controlled by the `Requirements` expression in the submit file, matches the policy expressed in the TSPC. It is expected that subsequent refinements will permit users to specify these restrictions in one place and have both policies match.

5.4 At a Worker Node

When a scheduler forwards a job to a WN, the delegation sequence will include service specific information as described in Section 4.5, so that the proxy credential used by the task when it executes identifies both the submitting user and the WN service credential. Before executing the task, a WN verifies that the task was intended to be run on this node, by evaluating the execute action expression in the TSPC. In addition, the WN is responsible for checking that the credentials it possesses identify the task it is executing, checking the signature on the task ClassAd, and verifying that any accompanying files are unaltered. If any of these checks fail, the job will be rejected by the WN before it is run.

Example configuration file entries controlling verification:

```
VERIFY_SIGNED_CLASSADS = True
VERIFY_JOB_CLASSAD_ATTRIBUTES =Owner, JobUniverse, CmdHash, CmdHashType, \
Arguments, ExecutionHostAAE
```

While the job runs, it has access to the credential chain including the SSPC, and it can use these credentials to authenticate to external sources such as a storage element or the `condor_credd`, to obtain the symmetric key used to decrypt the job. In the next section, we describe how the process retrieves its key from the `condor_credd`.

5.5 Accessing data using the SSPC chain

When a task running on a WN accesses the `condor_credd`, the `condor_credd` process evaluates both the execute action expression to confirm that the WN it is communicating with is authorized to execute the task, and the access expression to determine that the task is authorized by the user to access the secret. All of this information is available to

the `condor_credd` within the proxy delegation chain used by the user's process on the WN when it authenticates: the policies are contained in the task specific proxy certificate, and the information about the WN is included within the service specific proxy certificate.

If the authentication and authorization between the `condor_credd` and the job running on the WN are successful, the job (a simple wrapper script) is able to obtain the secret key used to encrypt the real job. The encrypted job is decrypted and runs.

The `condor_credd` stores the secret key in a hash table indexed by the SHA1 sum of the TSPC.

Although our implementation does not include this, the wrapper script could, upon job completion, encrypt the result using the same key.

5.6 Task Completion

When the task completes, a *receipt* is produced: the task ClassAd is signed, including cryptographic checksums on output files that are to be returned to the submitter. If service-specific proxy certificates are used, the signature here is made by the service-specific proxy credential and is performed by the starter process after the job runs. If they are not, the receipt is signed by the host certificate directly. This receipt and the results are returned along the task pathway as is usual within Condor. When the submitter obtains the results, they are able to confirm that the policies they specified were enforced by verifying the receipt: the receipt is signed by a service-specific proxy credential or host credential identifying a WN which satisfies the execute expression, and the signature is valid.

Chapter 6

Implementation

This chapter describes the implementation of the framework and gives a description of the underlying mechanisms and tools used by the components. In addition, an overview of the software design is presented.

The implementation and tests performed in the following chapter include each component of the framework. An example of how the framework components work together is presented in Chapter 5, showing how each of the components work together at the various places within the distributed system to provide end-to-end confidentiality for task input data and results.

The mechanisms were implemented entirely within the Condor code base, and rely on and take advantage of the APIs provided by the Globus Toolkit and OpenSSL, which are already linked with Condor. Essentially no code within these libraries was altered.[1]

6.1 Signed ClassAds

The following configuration file entries affect signing:

- `SIGN_CLASSADS`

  If set to "True", job ads within `condor_submit`, and machine ads within `condor_startd` are signed. In the implementation described in this thesis, other ad types are not signed but given the architecture and the way Condor uses the ClassAd library, it would be easy to add signatures to other ClassAd types.

[1] We found a bug in Globus Toolkit's handling of proxy policies, reported it to the Globus developers, and fixed it in the copy that we tested with.

- `VERIFY_CLASSADS`

  If set to "True", job ads are verified by `condor_starter` before jobs are executed.

- `CLASSAD_SIGNATURE_CREDENTIAL_TYPE`

  Determines the credential type for signed ClassAds. Valid entries are "SSL" and "GSI". With respect to signing ClassAds, these types are equivalent; this configuration option exists because so that the signing code knows which other configuration options to look for to find credentials to be used for signing.

- `AUTH_SSL_CLIENT_KEYFILE`

  `AUTH_SSL_SERVER_KEYFILE`

  `AUTH_SSL_CLIENT_CERTFILE`

  `AUTH_SSL_SERVER_CERTFILE`

  `AUTH_SSL_SERVER_CAFILE`

  `AUTH_SSL_CLIENT_CAFILE`

  Specifies the location for the SSL key file, certificate and trust root file when the SSL credential type is specified. Role (i.e. client, server) is determined from context.

- `GSI_DAEMON_KEY`

  `GSI_DAEMON_CERT`

  `GSI_DAEMON_TRUSTED_CA_DIR`

  Specifies the location for the GSI key file, certificate, and trust root directory when the GSI credential type is specified. When signing job ClassAds (i.e. within `condor_submit`), these attributes are ignored; instead the proxy credentials specified as `X509UserProxy` in the submit file are used instead (see below).

- `SIGN_[ClassAd type]_CLASSAD_ATTRIBUTES`

  When a ClassAd is signed, its type is determined and this configuration variable is consulted to determine which attributes should be included in the signature.

- `VERIFY_[ClassAd type]_CLASSAD_ATTRIBUTES`

  When a signed ClassAd is verified, its type is determined and this configuration variable is consulted to determine which attributes must be present in the signature and have identical values to those present in the ClassAd in order for the verification to return success.

For example, the following configuration file entries would used to sign and verify particular attributes in machine ClassAds:

```
SIGN_MACHINE_CLASSAD_ATTRIBUTES = Name, PublicNetworkIpAddr, CondorVersion, \
CondorPlatform, SlotID, VirtualMemory
VERIFY_MACHINE_CLASSAD_ATTRIBUTES = Name, PublicNetworkIpAddr, CondorVersion, \
CondorPlatform, SlotID, VirtualMemory
```

The following configuration file entries affect cryptographic checksumming of files; while this is independent of signing, when used with signed ClassAds, these checksums can be used to ensure end-to-end integrity.

- `CMD_HASH`

  When this attribute is set to true, a cryptographic checksum or hash is computed for the contents of the executable specified in the submit file by `condor_submit`. Note that at the time of writing, only the executable (not any other files transferred such as input files) is hashed; however, the Condor team has expressed interest in extending this functionality to all transferred files.

- `VERIFY_CMD_HASH`

  When this attribute is set to true, the cryptographic checksum identifying the executable is checked by `condor_starter` before a job is executed. To ensure end-to-end task integrity, set this configuration variable and include the "Cmd" attribute in the list of attributes to sign and verify.

- `CMD_HASH_TYPE`

  This attribute determines the hashing algorithm. This option is passed directly to OpenSSL's EVP interface. The default is "sha1".

The tool `condor_advertise` and `condor_sign` can be used to sign any ad type.

When `condor_advertise` and `condor_sign` are used, they ignore the `SIGN_CLASSADS` attribute above: The `condor_advertise` command line option `-sign` turns on signing in the tool (`condor_sign` always signs).

The following process is used to perform a signature:

1. The `SIGN_CLASSADS` configuration variable is consulted. If it is true, signing proceeds; otherwise the signature routine returns an error.

2. The ClassAd type is determined from the ClassAd and the list of attributes used in the signature are determined from the appropriate configuration variable (described above). If this configuration variable is not present, the signature routine returns an error.

3. The credential type is determined, and the signing key and associated certificate are read. In the case of GSI credential types, these are determined through the configuration file or the ClassAd itself; in the case of SSL credentials, they are determined by the configuration file. If the necessary files are not readable, the signature routine returns an error.

4. A new ClassAd object is produced which contains only the specified attributes. For those following closely at this level of detail, there is some special handling of the `Arguments` attribute, which is altered later in the submission process for (now unnecessary) backward compatibility reasons, and for caching relevant discarded attributes between invocations of `queue()` in `condor_submit`. This special handling only affects job ClassAds.

5. The text of the certificate for the credential used for signing is inserted into this ClassAd, along with a signature version. The attribute names are:
   - `ClassAdSignatureCertificate`
   - `ClassAdSignatureVersion`
6. This ClassAd object is canonicalized and serialized to a text string (the signed text).
7. The OpenSSL EVP interface for signing is applied to this text string with the key found above. If EVP fails for some reason (e.g. a corrupted key) the signature routine returns an error.
8. The resulting binary string is transformed into a text string (the signature string).
9. The serialized ClassAd text and the signature string is added to the original ClassAd in the `ClassAdSignatureText` and `ClassAdSignature` attributes, respectively.

The following process is used to verify a signature:

1. The `VERIFY_CLASSADS` configuration variable is consulted. If it is true, verification proceeds; otherwise the verification routine returns an error.
2. The ClassAd type is determined from the contents of the signed ClassAd, and the appropriate configuration variable is consulted to determine which attributes should be used to verify the ClassAd. Note that the list of attributes to verify may differ from the list of attributes used at signature time; in particular, a subset of signed attributes may be used for verification.
3. The signed text and the signature are extracted from the appropriate attributes in the signed ClassAd: `ClassAdSignatureText` and `ClassAdSignature`, respectively. If these attributes are not present, the verification routine returns an error.
4. The text signature is transformed into a binary representation; this reverses a similar step in the signature process.

5. The signed text is deserialized into a ClassAd.

6. The `ClassAdSignatureCertificate` and `ClassAdSignatureVersion` are extracted from the certificate. At present, the only signature version that is supported is "1.0a". If an unsupported signature version is present, the verification routine returns an error. The certificate extracted here will be used to actually verify the certificate.

7. The certificate is checked against the trust roots defined in the configuration file entries (`GSI_DAEMON_TRUSTED_CA_DIR` or `AUTH_SSL_[CLIENT|SERVER]_CAFILE`). If the certificate was not issued by one of the trust roots, the verification routine returns an error.

8. The OpenSSL EVP interface for verifying is applied to the text string with the binary signature string and the public key from the certificate. If EVP fails, the verification routine returns an error.

9. The list of attributes to verify from step 2 above is consulted and an error is returned if any of the attributes are either not present in the signed ClassAd or differ from the corresponding attributes in the signed text.

Some practical considerations result from this signing mechanism:

1. The total size of the ClassAd increases; it will at most double in size (every attribute in the ClassAd is included in the signature as well) and increase by a constant factor (because the text of the certificate is included).

2. If a signature verification fails because an attribute required for verification is altered but the signature verifies at the cryptographic level, it is possible to determine the value of the attribute at the time of signing. The present implementation does not make use of this but it would be possible to extend the implementation to do so.

6.2 Task-Specific Proxy Credentials

The following configuration file entries affect task-specific proxy credentials:

- `ADD_TASK_POLICY`

  If this configuration variable is set to true, when a job is submitted by `condor_submit`, task-specific proxy credentials are generated and used as described below.

- `CHECK_TASK_POLICY`

  If this configuration variable is set to true, when a job is executed by `condor_starter`, task-specific proxy credentials are checked as described below.

The following process is used to create a task-specific proxy credential:

1. The task policy text is assembled after signing by concatenating attributes from the job as follows: The job attribute `ClassAdSignatureText`, a semicolon (";"), and the job attribute`ClassAdSignature`.

2. A "self-delegation" step is performed within the `condor_submit` process, using the Globus GSI wrapper for the underlying OpenSSL cryptographic libraries. There is a potential for a performance optimization here: the self-delegation step is not required by the design or by the API for TSPC (although it is required by the API for SSPC as described below).

   The `condor_submit` process generates a keypair and follows the standard delegation protocol steps with the following steps: the policy is included, and the certificate type coerced to `GLOBUS_GSI_CERT_UTILS_TYPE_RFC_RESTRICTED_PROXY`.

The following process is used to authorize a task-specific proxy credential:

1. The job proxy certificate chain is authenticated normally, and the TSPC policy field is identified by the policy callback. The authorization step (verifying that the job ad signature is valid and identical to the policy) is not performed as part of the authentication step, so this callback returns immediately.

2. The job ClassAd signature is verified as described above.

3. The expected policy string is assembled by concatenating the `ClassAdSignatureText` attribute, a semicolon (";"), and the `ClassAdSignature` attribute.

4. The policy is extracted from the certificate chain. There must be exactly one policy in the chain.

5. The expected policy string is compared with the actual policy. It must match exactly.

6.3 Action Authorization Expressions

Action authorization expressions (AAEs) are implemented as expressions in the ClassAd language. At present, there are no configuration file entries affecting AAEs; the user directly includes expressions in their submit file:

```
+ExecutionHostAAE = ...
```

The following action authorization expressions are evaluated in the `condor_credd` process when a shared secret is retrieved (after the service-specific proxy credential is authenticated):

- `ExecutionHostAAE`: This expression must evaluate to true and may depend on these attributes supplied by the `condor_credd` process based on the contents of the SSPC:
  - `ExecutionHost`: This is set to the distinguished name (DN) from the service certificate of the process which issued the SSPC. Note that this service certificate's certificate chain is checked for validity and must originate with a CA root certificate explicitly trusted by the `condor_credd` process.
  - `ExecutionHostIssuer`: This is set to the DN of the issuer of the service certificate.
- `AccessAAE`: This expression must evaluate to true and may depend on the `CreddAccess` attribute which is set by the `condor_credd` process to "get".

The fields implemented here are a subset of the fields that could be extracted from the service-specific proxy certificate or populated by the `condor_credd`. Additional fields could easily be defined as use cases arise.

6.4 Service-Specific Proxy Credentials

The service-specific proxy credential is a regular proxy certificate with additional information in the policy field. This additional information takes the form of a signed ClassAd.

The policy ClassAd contains the following attributes:

- `ProxyPublicKey`: The public key of the proxy containing the policy.
- `Assertion`: The text of the assertion is: "Private key associated with ProxyPublicKey is present on signing host."
- `HostCertificate`: The certificate chain of the signing host.

This ClassAd is signed using the private key of the WN on which the job is to be run. This private key corresponds to the one in the `HostCertificate` field.

To simplify the implementation, this ClassAd is re-delegated on the WN: instead of being assembled and signed by the proxy certificate on the scheduler preceding the WN in the delegation chain, a normal delegation step occurs between the scheduler and the WN, followed by a re-delegation step on the WN.

In order to perform the authentication step for an SSPC chain, the certificate chain and policy it contains is converted after verification into a certificate chain object. In particular, this object contains the SSPC policy and the TSPC policy.

A number of steps occur to check the validity of the credential chain:

- The chain is checked for validity.
- The job ClassAd signature is checked as described above.
- The TSPC is checked as described above.
- The SSPC is checked as follows.

To check the SSPC, the verifier:

- Extracts the policy from the SSPC. This policy is a signed ClassAd we'll call the Host Assertion ClassAd (HAC).
- Checks the signature on the HAC.
- Verifies that the public key in the `ProxyPublicKey` field is the same as the enclosing proxy certificate.
- Checks the certificate chain on the `HostCertificate`.
- Matches the `HostCertificate` to the signature on the HAC.
- Checks that the assertion is the appropriate string: "Private key associated with ProxyPublicKey is present on signing host."

6.5 Confidential Data and the Secure Key Storage Service

Given the infrastructure described above, the basic implementation of confidential tasks is relatively straightforward.

- The confidential data is encrypted by the user before the job is submitted.
- The job itself is a wrapper script that first obtains the key from the `condor_credd` using the SSPC.
- The job then decrypts the encrypted data and runs the "real" job.
- After completing the "real" job, the wrapper encrypts any sensitive data before exiting.

The secure key storage service, the `condor_credd` is a simple daemon that verifies TSPC and SSPC certificate chains. Keys are stored in a simple hash table indexed by the SHA1 sum of the TSPC. Only SSPC credential chains with valid SSPC credentials are authorized to attempt to get secret keys. Instead of requesting a secret key by index directly, the requester just authenticates and requests a secret key. The `condor_credd` indexes by calculating the SHA1 sum of the TSPC in the credential chain, and returns the

Chapter 7

Performance Analysis

7.1 Introduction

For the security mechanisms described in this thesis to be useful, they must have a minimal negative (if not positive) effect on performance, and their performance characteristics must be well understood. To show how the security mechanisms affect the performance of Condor, we measure and compare the performance of Condor with and without the mechanisms in place, vary the combinations of mechanisms, and vary the configuration of Condor and the tasks we submit in testing. We conclude that the performance effect of these mechanisms can be positive, and when negative, is generally small.

Since the mechanisms are based on established cryptographic techniques such as digital signatures, cryptographic hashing and encryption based on well studied standards such as RSA, AES, MD5, and X.509, the analysis is based on measuring actual performance of the Condor system submitting and running real jobs. Unsurprisingly, mechanisms based on digital signatures, hashing and encryption scale in a manner that is roughly linear in the size of the input; this is unsurprising because this is the behavior of the underlying cryptographic primitives. The point of the analysis is to provide engineers with guidelines for forecasting the actual performance impact of adopting these mechanisms in working systems relative to the other components of the working system.

The performance analysis described in this chapter consists of three distinct sets of comparisons.

In the first, the operations of the components of Condor into which the security mechanisms are integrated are measured in detail. The operations of the security mechanisms are isolated from the rest of the operations of the daemons, and the amount of time each stage takes is shown while varying job characteristics, security mechanism settings, and framework components. This permits a direct comparison of performance of the framework and non-framework operations and how both are affected by various changes. In addition to showing the performance of the framework within each component, this comparison shows exactly where in the system the security mechanisms are implemented.

In the second, key characteristics of the workload, input size and job runtime, are varied and the comparison is performed between the range of security framework options. The key measurement here is the total amount of time spent processing the job from start to finish (the time spent in queueing is omitted). This permits, for example, a direct comparison between the time spent processing a 30 second job with 1 MB input data with full confidentiality turned on vs. just signatures.

In the third, the performance of a component central to the scalability of the Condor infrastructure, `condor_collector`, is measured under load. This component can become a bottleneck in certain situations. Two approaches to addressing this bottleneck are described, one based on signed ClassAds, and the advantages and disadvantages of each approach, including performance, are discussed.

7.2 Phases of Execution

To measure performance, we added timestamps using `gettimeofday` to the Condor code at the points where our security mechanisms are implemented. To minimize skew resulting from the overhead of making this system call, we ensure that whenever we directly compare a span of time between two different configurations, in each span, the same number of calls to `gettimeofday` are performed. Each measurement we show is averaged over a large number of invocations. Performance was measured on a Linux system with an Intel Core 2 Duo CPU and Seagate ST3500630AS disk, running 64-bit CentOS 5.3.

Timestamps were recorded at the beginning and ends of the following spans in (and associated with) `condor_submit`:

1. from the invocation of `condor_submit` to when the process exits, including:
   (a) *when encryption is turned on, a wrapper for* `condor_submit` *encrypts the input data*
   (b) `condor_submit` authenticates to the scheduler and obtain a job identifier
   (c) *hashing of the executable and other input data to be transferred*
   (d) *signing the job classad*
   (e) *creating the task-specific proxy certificate (TSPC)*
   (f) *preparing the secret*
   (g) *authenticating before transmitting the secret to the* `condor_credd`
   (h) *completing the transaction with the* `condor_credd`
   (i) *authenticating to* `condor_submit` *with the TSPC*
   (j) *when proxy delegation occurs between* `condor_submit` *and* `condor_schedd`

The following figures show the run times of the stages described above for `condor_submit`. In Figure 7.1, the timing for both framework and non-framework stages are shown comparing three different executables of different sizes. The bar graph at the top permits direct comparison of each of the stages; note that the hashing phase in particular increases roughly linearly with the size of the input and that the time taken to perform the framework stages is not large compared to the other stages. The horizontal stacked bar graph on the bottom shows a different perspective on the same data; the stages are shown laid end-to-end.

In Figure 7.2, the same chart type is used to show how the lengths of the stages vary when different authentication methods are used.

Figure 7.3 shows how the lengths of the stages vary when different framework mechanisms are employed.

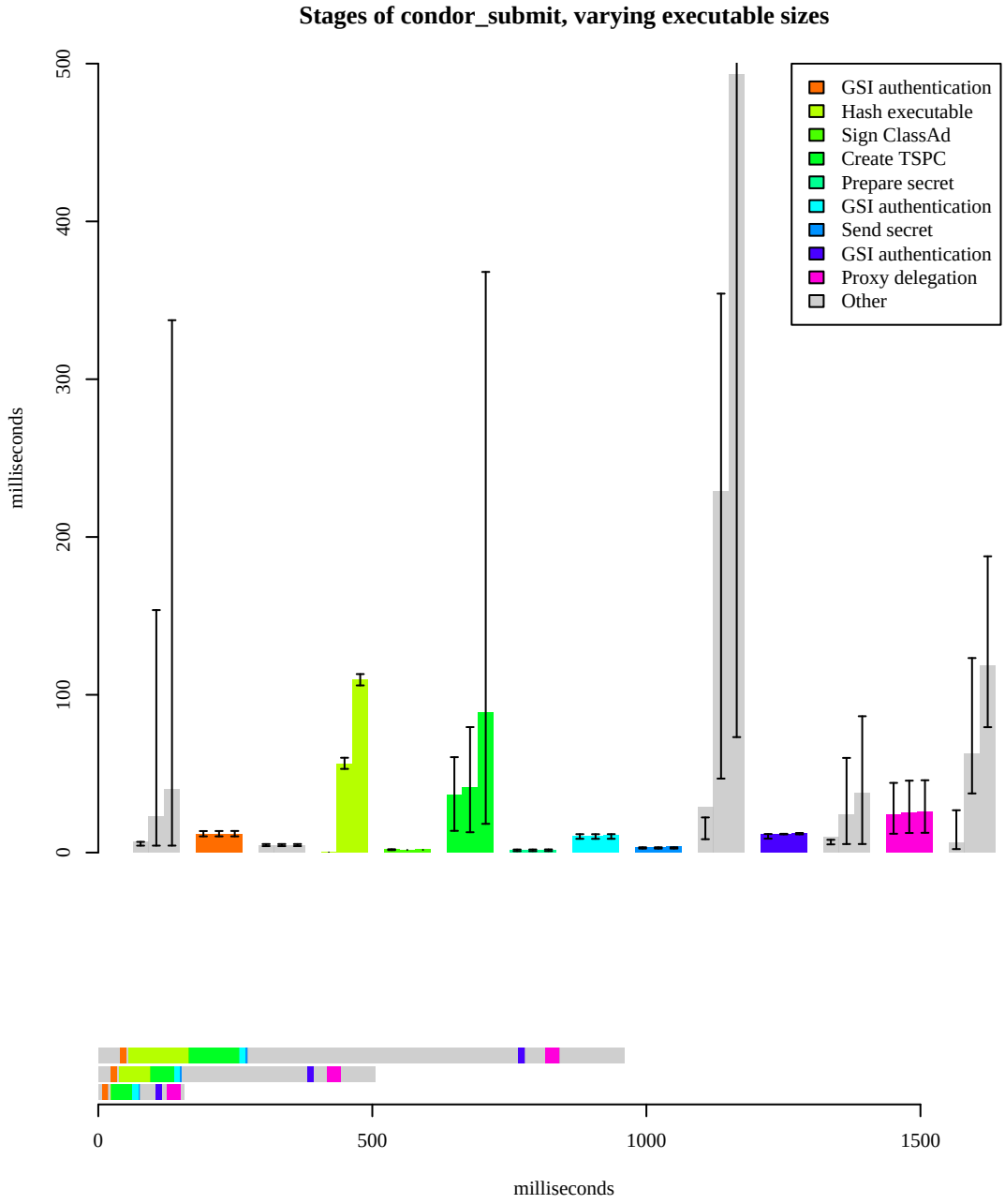

Figure 7.1 **Stages of `condor_submit`, comparing executable sizes.** Three different executable sizes are shown. The first executable, leftmost on the upper chart and lowest on the lower chart, is a small script, 60 bytes. The second is a 12 MB binary, and the third is a 24 MB binary

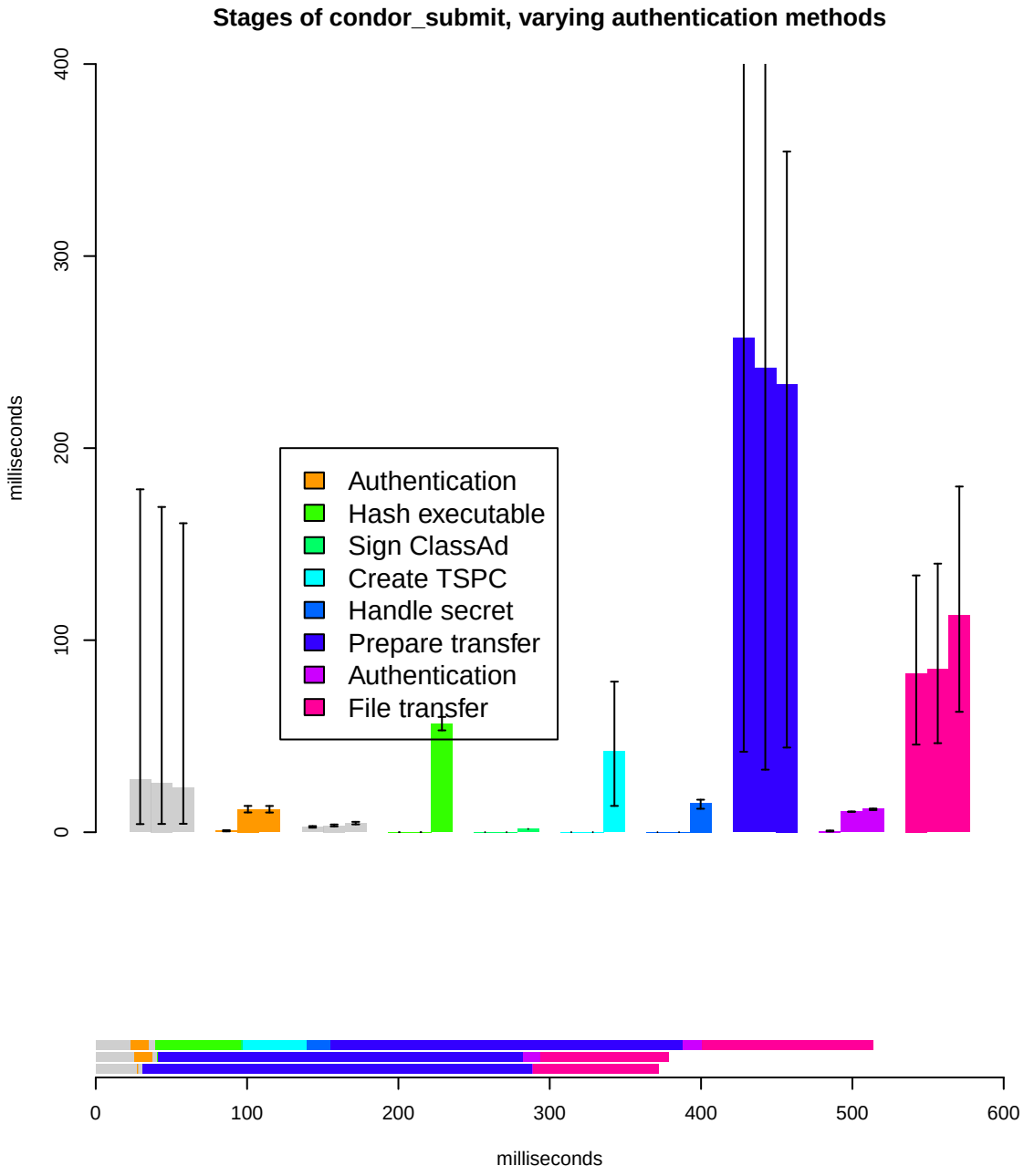

Figure 7.2 **Stages of `condor_submit`, differing authentication methods.** Three different authentication methods are shown. The methods compared are password, GSI, and the full framework including TSPC and secret sharing

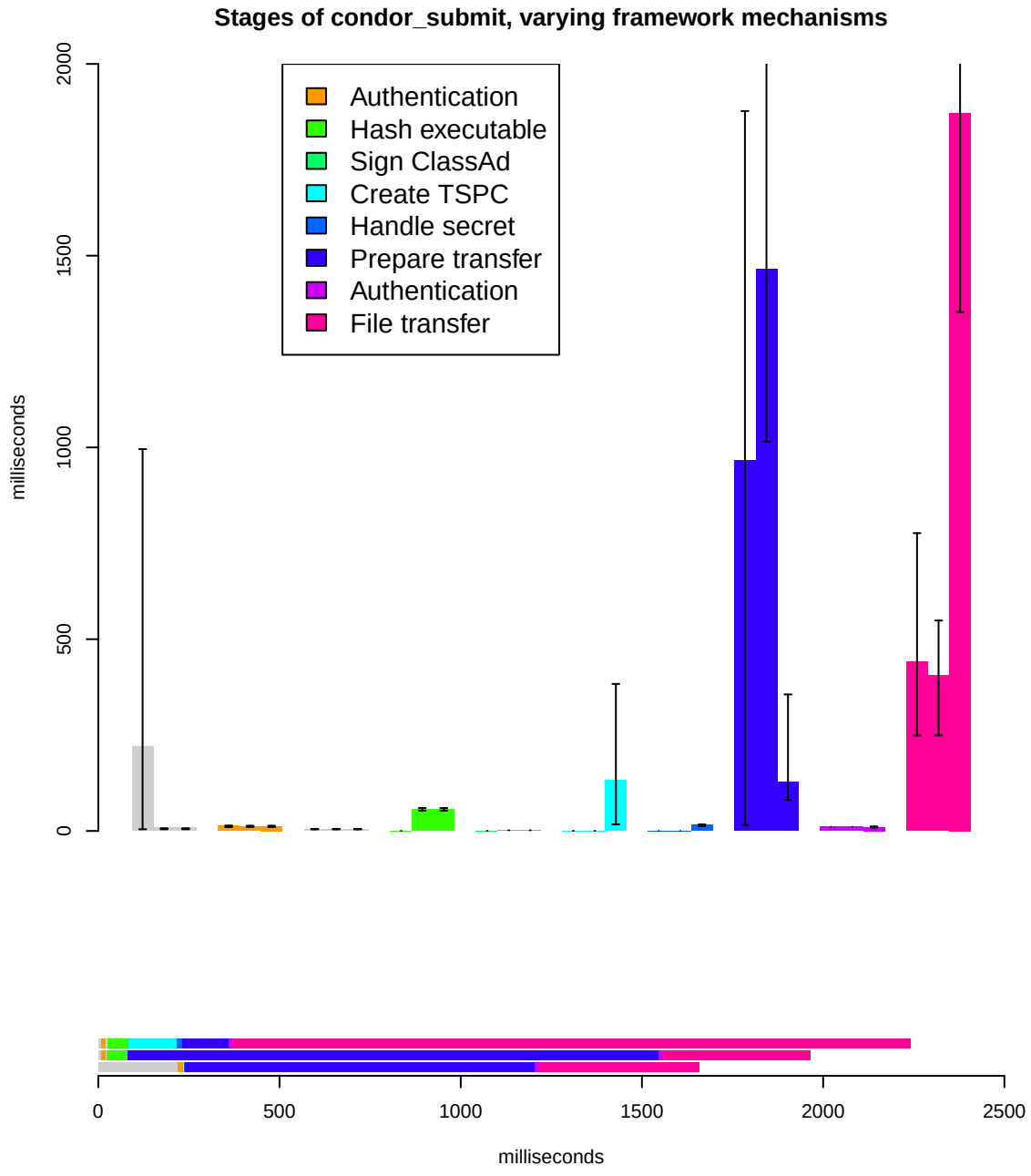

Figure 7.3 **Stages of `condor_submit`, differing framework components.** Three different framework components are shown. Hashing and signing is shown leftmost and lowest, followed by TSPC and then full secret sharing to support encryption.

Within `condor_starter`, timestamps were recorded at the beginning and end of the following spans:

1. from the invocation of `condor_starter` to when the process exits, including:
   (a) *verifying the signed ClassAd*
   (b) *checking the TSPC*
   (c) running the job
   (d) checksumming the output data
   (e) signing the receipt

Within `condor_credd`, timestamps were recorded at the beginning and end of the following spans:

1. the *store* session, including:
   (a) authenticating to begin the *store* session
   (b) the full *store* session: storing the TSPC and the secret
2. the *get* session, including:
   (a) authenticating to begin the *get* session
   (b) checking the SSPC
   (c) evaluating the AAE

7.3 The Effect of Job Run Time and Input Size

The previous section shows where within Condor overhead from the security framework occurs. This section quantifies the overhead relative to the factors that affect it most: the framework mechanisms used, the amount of data involved, and the duration of the computation performed on the data.

We compute the overhead by calculating the time it takes to submit and start jobs and the portion of time spent in the spans described above which comprise the the framework mechanisms. The overhead is calculated as the portion of the total spent in framework mechanisms.

To show the effect of a large range of data sizes in Figure 7.4, we measured the overhead when transferring input data from 1 B to 512 MB, doubling the amount each time, so that the X axis is log-scale. Each of the six graphs shows a different job run time. The log-scale X axis shows the amount of input data (including the executable) for the computation and the Y axis shows the percentage overhead for each of the framework components. Note that the run time was simulated and idle run time added after the fact rather than actually executed since it would have no meaningful effect on the results.

7.4 Authenticated Messages vs. Authenticated Sessions

One of the crucial features of the existing security mechanisms in Condor is that authentication for communications is session based. When Condor daemons and tools authenticate, they exchange a session key which is used for the rest of the communication between the daemons. This is advantageous in situations where communications between daemons involves transferring a large quantity of data, or is likely to resume quickly.

In contrast, authenticated messages are useful when there are many different sources of messages each of which only needs to send a small number of messages. Instead of authenticating, exchanging a session key, and then using the session key to provide integrity guarantees, each message is individually signed, permitting the recipient to identify the sender.

The `condor_collector` daemon is one which receives a messages from a large number of distinct sources: each machine in a Condor pool sends a message to the collector on a configurable periodic basis. The overhead of authenticating and establishing session keys limits the rate at which new messages can be received by the collector, and so can limit the size of the pools.

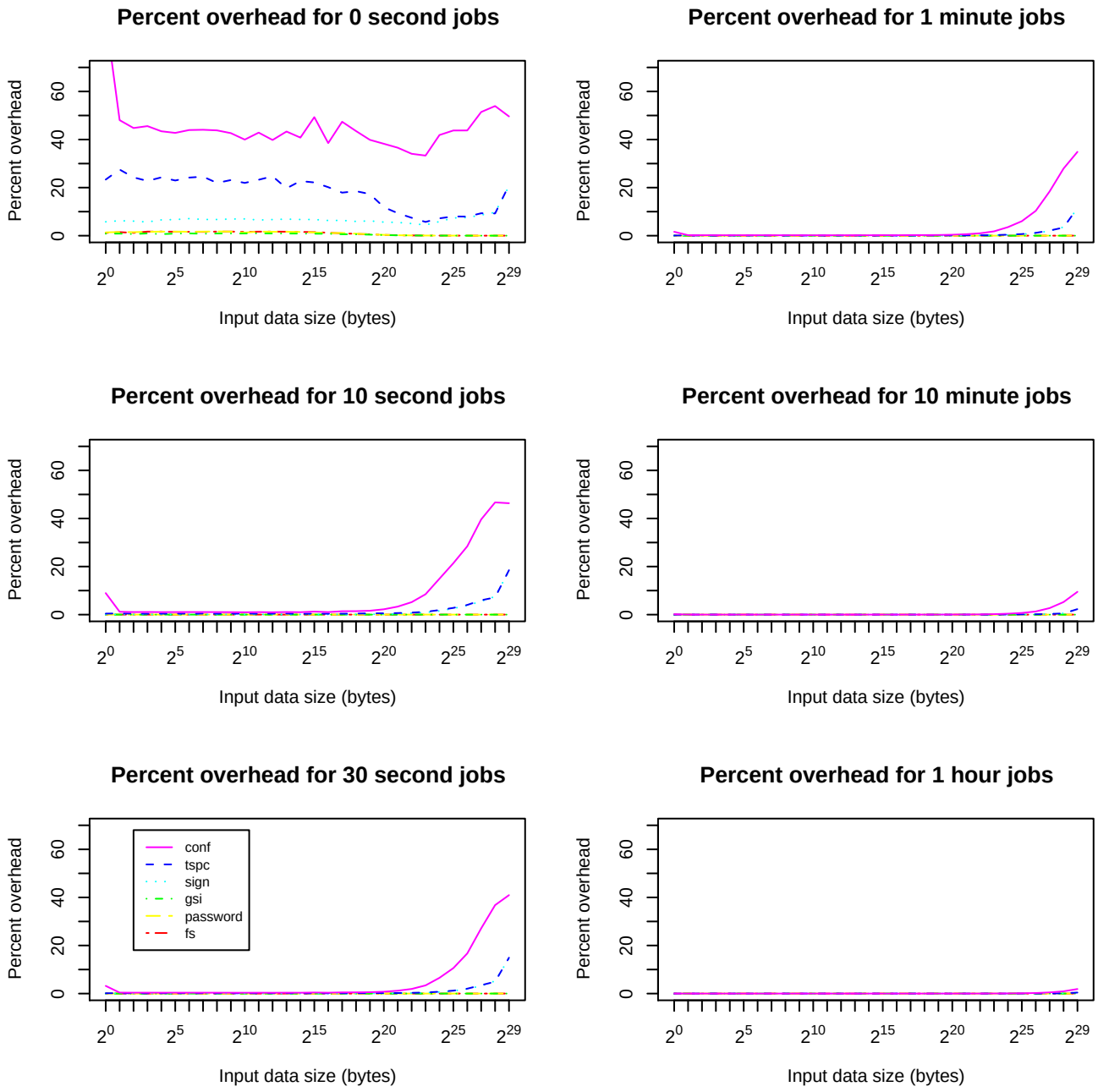

Figure 7.4 **Relative overhead.** The relative overhead for a range of input data sizes and job run-times.

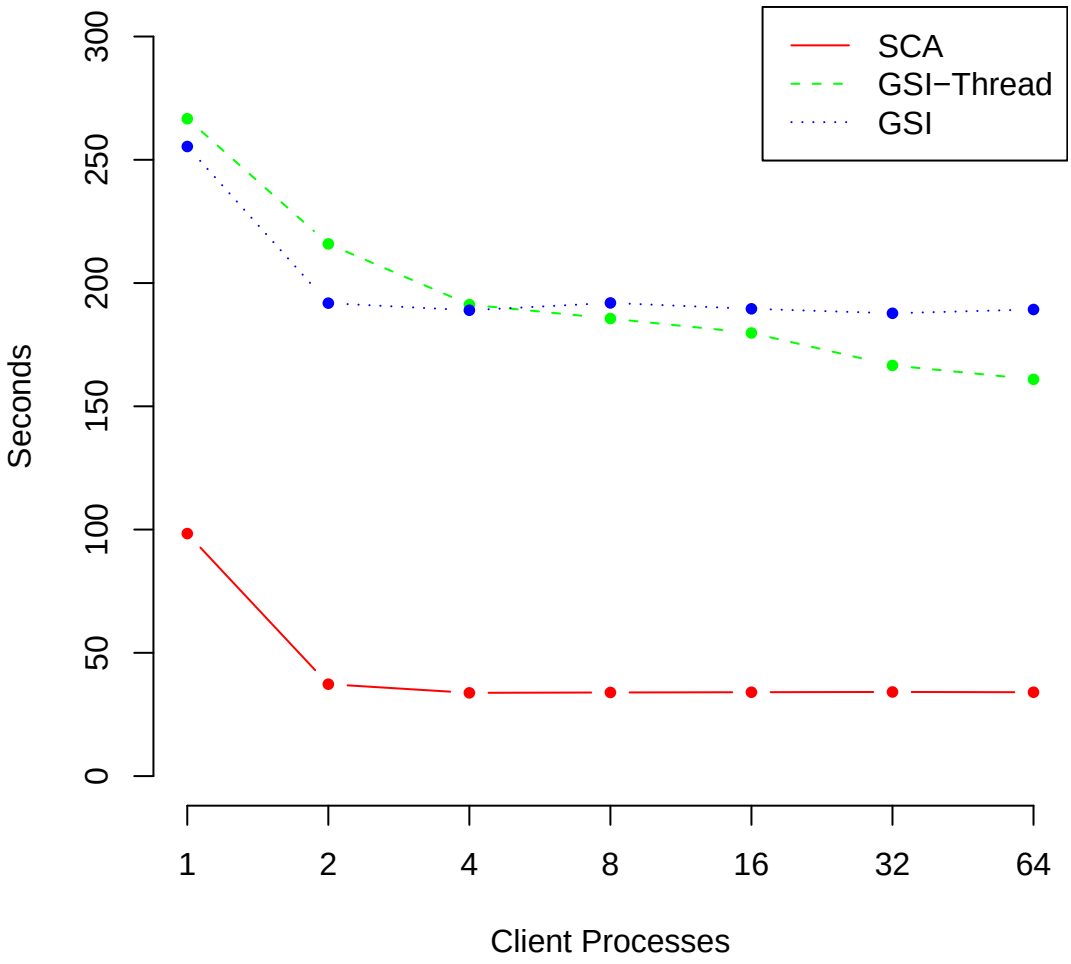

Figure 7.5 **Ingestion rates.** The amount of time required for the collector to accept 16384 distinct incoming ClassAds

Signed ClassAds can be used as authenticated messages. Figure 7.5 shows the rates at which the collector can process incoming messages using authenticated sessions vs authenticated messages. The X-axis shows the number of client processes supplying ClassAds to the collector daemon. The Y-axis shows the amount of time between when the processes start and when they complete. Three configurations are compared: SCA, indicating ClassAds that have an encrypted component and are signed, but for which there is no authenticated session communication, GSI, indicating standard GSI authentication including a session key providing encryption and integrity, and GSI-Thread, in which the same GSI authentication protocol is employed but multiple threads are used.

Chapter 8

Security Requirements Analysis

8.1 Introduction

This section analyzes the security of existing distributed batch computing systems by describing the security requirements of these systems in detail, and comparing the extent to which a variety of existing systems meet these requirements.

8.1.1 Security Principles and Terminology

This section describes some basic security terminology and gives examples of how these terms relate to concrete topics in distributed batch computing. In particular, the term *security requirement* is defined in terms of other terms that are more clearly defined in security textbooks.

The operators of a particular system define a policy regarding who can access the system. To achieve these policy goals they use various *mechanisms*. *Attackers* have incentives to prevent these goals from being achieved. *Vulnerabilities* are ways that attackers can circumvent the security mechanisms.

Often, the goals that a particular security mechanism is intended to achieve can be categorized into the overlapping goal areas *availability*, *confidentiality* and *integrity*.The property of availability is whether or not a particular piece of data can be accessed. So, a mechanism is said to protect availability if the mechanism can be used to prevent an attack that would limit a valid user's attempt to access data he is permitted to access. Similarly, a system can be said to have a vulnerability affecting confidentiality if a user not permitted to read

a particular piece of data can do so. Mechanisms protecting integrity are intended to keep data from being altered.

By tradition, the mechanisms that can be used to achieve these goals are *authentication*, *authorization*, and *audit*. Authentication is the process of determining *who* a particular user or process is; Authorization is the process of determining *what* a particular user or process is permitted to do. Authentication and authorization mechanisms can be separate although they are often combined: a user who can authenticate to the system may be authorized to do some basic operations, but other operations require additional authorization.

Audit mechanisms behave differently in that they exist so that it is possible to determine what has happened. For example, if a piece of data has been altered, mechanisms protecting the integrity of the data have failed. Audit mechanisms provide the system with a way of determining that the integrity has been violated, so it will not be mistaken for valid data. In distributed systems security, a great deal of attention has been paid to authentication and authorization; this thesis emphasizes the value of audit mechanisms.

The *security requirements* of a system are the set of security policy goals that operators of a given system might want the system implementation to support through security mechanisms. The requirements do not specify how these policies should be implemented, only that they are likely to be the policy goals of some system operator.

8.1.2 Anonymity, Privacy and Confidentiality

The concepts of anonymity, privacy and confidentiality are related and often confused, since they can have different definitions in different contexts. In the context of distributed batch computing systems, we use the following definitions

Anonymity means that a user who submits a job can not be distinctly identified. The user may have authenticated to some entity, and carry some credential indicating that they have authenticated but as long as the anonymity property holds, the user can not be distinctly identified.

Privacy means that a user who submits a job can not be distinctly identified except by the authentication system they use to enter the system. They may carry a credential; only their local authentication system can identify them given this credential.

Finally, *confidentiality* refers to the secrecy of data that is part of a workflow, not the identity of the users who submitted that workflow.

8.2 Requirements

This section describes requirements that are relevant to security within the scope of a distributed batch computing environment, particularly one which spans multiple administrative domains.

The following requirements are paraphrased from requirements detailed in Broadfoot and Lowe [25]:

1. **Dynamic network**: Users, machines, organizations and administrative domains must be able to come and go from the environment.

2. **Single sign-on**: Users must be able to authenticate to the system once rather than re-authenticating with each resource or intermediary they interact with.

3. **Interoperation**: Components of the system must be able to interoperate with different organizations that have differing policies.

4. **Scalability**: The system must support large numbers of users, jobs and resources.

5. **Restricted delegation**: Users must be able to delegate their access rights to processes that run on their behalf. Furthermore, users must be able to restrict the access rights that are delegated.

6. **Separation of authentication from authorization**: Authentication mechanisms must be separate from authorization mechanisms because they may occur in different places at different times.

7. **Revocation**: Administrators must be able to revoke user credentials and change access control policies (both authorization and authentication); In some cases this revocation must be instant.

8. **Confidentiality**: Users may have data that they require must not be revealed to other users or resource or intermediary owners within the system.

9. **Privacy**: Some users or resources may require that the system not reveal their identity to other components processing the workflow.

10. **Trust relationships**: Trust relationships must be explicit to support delegation.

11. **Integrity**: Integrity of data and code "must be beyond reproach".

12. **Integrity of delegated rights**: In addition to integrity of data and code, rights associated with delegated credentials must not be alterable.

13. **Non-repudiation**: Users must be able to determine that a resource agreed to run, or ran, their jobs. Users and resources must be able to determine their rights and responsibilities with respect to workflow transactions. [1]

14. **Freshness**: System components must be able to determine credential freshness, especially with respect to authentication and authorization.

In addition to these requirements, I propose the following requirements:

15. **Task-specific credentials**: Users should be able to link delegated credentials with a particular task so that the credentials may not be used for other tasks.

16. **User-specified access control**: Users need ways to express policy requirements about where their jobs run and how they are handled.

[1] "Non-repudiation is concerned with the ability to prove that a given entity performed or agreed to a particular task, even in the face of denial. This is particularly important in e-commerce where money transactions take place. This is not a security aspect that has been considered in any detail within the grid, but it will become important as the grid technology matures and is used by applications involving financial exchanges. If accessing resources starts costing money, then both parties must be assured that the other fulfills their duty; in cases where one party fails, proof of commitment is necessary to formally resolve the dispute." [25]

17. **Attribution and reproducibility**: When the validity of some workflow results are challenged (i.e. because a host or site are found to have been compromised), it may be necessary to recalculate a subset of the results from the same inputs and verify that the same results were produced.

18. **Minimize trust**: The number of processes that users and resources must trust should be kept to a minimum.

8.3 Delegation Frameworks

This section describes the implementation of security mechanisms in several delegation frameworks. This list is in no way exhaustive: the focus of study here are systems that support delegation chaining and thus appropriate for environments with multiple administrative domains.

8.3.1 Delegation Chaining in DSSA

Gasser and McDermott introduced delegation chaining in the context of Digital's Distributed System Security Architecture (DSSA) [43]. Along with Neuman's work on restricted proxy delegation [72], this work pioneered certificate-based delegation systems that are common today. Delegation chains are formed by repeated applications of the following delegation protocol.

8.3.1.1 Delegation Protocol

In the following protocol, the *delegator* host H_n, in possession of a valid certificate associated with credential C_n (perhaps a proxy certificate), representing user U, extends the delegation chain to a *delegate* host H_{n+1} by issuing a proxy certificate. Note that this protocol is very similar to that used to form (non proxy) certificate chains as described in Section 3.5.1.

1. Hosts H_n and H_{n+1} mutually authenticate, using their long-term public key certificates, and establish a secure channel.

2. Host H_{n+1} creates a keypair C_{n+1} and retains the private portion. The public portion is sent to H_n.

3. H_n uses the private portion of C_n to sign a certificate that states that "C_{n+1} as H_{n+1} for U until T" (where T is a time period). H_n then transfers this certificate to H_{n+1}.

If user U's certificate is assumed to be C_0, the following chain is formed (in the notation used here, $X : Y$ is interpreted to mean that X signs statement Y):

$$C_0(U) : C_1 \text{ as } H_1 \text{ for } U \text{ until } T_1$$

$$C_1 : C_2 \text{ as } H_2 \text{ for } U \text{ until } T_2$$

$$C_2 : C_3 \text{ as } H_3 \text{ for } U \text{ until } T_3$$

$$\ldots$$

$$C_{n-1} : C_n \text{ as } H_n \text{ for } U \text{ until } T_n$$

In order to authenticate as a delegate of user U, host H_n must present certificates $C_0 \ldots C_n$, and demonstrate that it possesses the private portion of C_n.

8.3.1.2 Roles and Restricted Delegation

In addition to the basic delegation protocol, DSSA included a number of refinements intended to make it more practical and secure. In particular, they define ways that their mechanisms could be extended to include support for *roles* and *restricted delegation*.

Roles in DSSA were simple groups and worked in concert with ACLs stored with the objects credentials accessed. As a member of a group, a user could specify that a delegated

credential's access should be limited to the objects accessible by the group rather than the complete set of rights granted to the user. This increases the burden on the verifier who must now determine group membership. Many details of this mechanism were left unspecified.

While Gasser and McDermott argue that roles could meet much of the need for restricted delegation, they discuss other ways that restricted delegation might be implemented but conclude that those are impractical given the other characteristics of their system.

8.3.2 Foster

Foster [39] described an early version of GSI which did not support delegation chains and instead involved direct communication between the *user proxy* and the *resource proxy*, processes that ran in different administrative domains and serve to convey and convert credentials from one administrative domain to another. As a result, this approach permits delegation but is limited to only one effective level of delegation. The user proxy uses a delegated credential signed by the user. When a resource request is issued, the user proxy uses this credential to authenticate to the resource proxy to obtain access to the remote resource. If additional services (i.e. access to a second resource) are required by the remote process, the resource proxy communicates with the user proxy to request that the latter authenticate directly with the second resource. The user proxy generates a credential that the first resource can use to access the second.

Foster, et. al. note that this limits scalability, but argue that this limitation is required in order to permit restricted delegation and prevent user credentials being compromised if a security breach occurs on a resource.

8.3.3 GSI

Despite this, GSI was extended to support delegation chains [103, 104], and implemented in the context of the Globus Toolkit. The delegation protocol is substantially the same as that described by Gasser and McDermott, the key difference being that GSI is based on X.509 certificates and standardized [104]. To enable restricted delegation, the format for

proxy certificates was extended to include fields specifying a policy language identifier and a policy. However, no specific policy language was defined. In practice, the limited lifetime typically given to proxy certificates acts as the main limit on the usefulness of delegated credentials.

8.3.4 Condor

Condor (without the additional mechanisms provided by the framework we describe in this thesis) includes support for delegation chains through the GSI libraries provided by the Globus Toolkit. In addition to GSI, Condor includes support for a variety of other authentication mechanisms including Kerberos, but delegation is only supported when using GSI. Condor's architecture including security mechanisms is described in Chapter 3.

8.3.5 Our Framework

Our framework extends the delegation mechanisms present in GSI (and Condor) with the aim of reducing the scope of trust required of users and resources.

8.4 Analysis

In this section, we discuss how mechanisms to meet each of the security requirements described in Section 8.2 are implemented in the delegation frameworks described in Section 8.3. This information is summarized in Table 8.1. The description of each requirement is repeated here for readability.

1. **Dynamic network**: *Users, machines, organizations and administrative domains must be able to come and go from the environment.*

   Each of the frameworks permit users and machines to come and go, however, in the Gasser, et.al. framework it's not clear how multiple administrative domains are supported. Frameworks based on X.509 (all the others) support multiple administrative domains by having each host and user explicitly list a set of X.509 trust roots.

	Gasser	Foster	GSI / Condor	Condor + Framework
Dynamic Network	N/A	×	×	×
SSO	×	×	×	×
Interoperation	N/A	×	×	×
Scalability	N/A	–	×	×
Restricted Delegation	–	–	–	×
Auth-N v. Auth-Z	×	×	×	×
Revocation	×	–	–	–
Confidentiality	–	–	–	×
Privacy	–	–	–	–
Explicit Trust	×	×	–	N/A
Integrity	–	–	–	×
Unalterable Rights	×	×	×	×
Non-repudiation	–	–	–	×
Freshness	×	×	×	×
Task-specific credentials	–	–	–	×
User policies	–	–	–	×
Reproducibility	–	–	–	×
Minimize Trust	–	–	–	×

Table 8.1 **Security requirements.** A comparison of delegation frameworks showing how they can satisfy security requirements.

2. **Single sign-on**: *Users must be able to authenticate to the system once rather than re-authenticating with each resource or intermediary they interact with.*

   This is a critical requirement for all of the frameworks.

3. **Interoperation**: *Components of the system must be able to interoperate with different organizations that have differing policies.*

   Again, it is not clear to what extent the Gasser, et.al. was implemented to support multiple administrative domains.

4. **Scalability**: *The system must support large numbers of users, jobs and resources.*

   The scalability of the Gasser, et.al. framework is unclear. The role of the directory service is not made entirely clear, and the focus of this work is not on batch computing. In addition, the relative computational cost of cryptographic operations on modern hardware invalidates to some degree the discussion of scalability in this paper.

   Foster, et.al. explicitly describe scalability as one of the issues with their delegation scheme, due to the requirement that the user proxy handle delegation requests on behalf of resources, in order to facilitate restricted delegation and limit the consequence of security breaches revealing delegated credentials.

5. **Restricted delegation**: *Users must be able to delegate their access rights to processes that run on their behalf. Furthermore, users must be able to restrict the access rights that are delegated.*

   Delegation is a fundamental requirement for all of the systems.

   Gasser, et.al. discuss restricted delegation and reject it in favor of roles: users who are members of groups can limit their delegated credentials to the subset of their rights that are held by a group of which they are a member.

   Foster, et.al. implement restricted delegation by limiting delegation chains and placing the burden of determining whether a resource request should be satisfied on the user

proxy process. It is not clear how policy about restricted delegation is specified or enforced other than that the "decision" about whether to grant access is the responsibility of the user proxy.

While restricted delegation is permitted in the GSI framework, there is no standard policy language and in practice this mechanism is not used. For example, users of Condor have no way of specifying policy restricting delegation.

In our framework, the delegated credentials are tied to a particular task via task-specific proxy certificates and restrictions can be expressed using action authorization expressions in the ClassAd language. Although the range of primitives available in these expressions is somewhat limited, this range could easily be expanded as needed in future releases.

6. **Separation of authentication from authorization**: *Authentication mechanisms must be separate from authorization mechanisms because they may occur in different places at different times.*

   In each of the systems authentication is separated from authorization.

   In Condor and GSI, efforts to standardize callouts to external authorization mechanisms have been made and partially implemented, for example [41, 42].

7. **Revocation**: *Administrators must be able to revoke user credentials and change access control policies (both authorization and authentication); In some cases this revocation must be instant.*

   Gasser, et.al. include a facility to permit users to revoke credentials immediately by having each process involved in the delegation chain delete any associated private keys. This may not meet all needs, however, since credentials must be revoked *before* any of the systems holding delegated credentials are compromised.

8. **Confidentiality**: *Users may have data that they require must not be revealed to other users or resource or intermediary owners within the system.*

Short of Trusted Computing (see Section 9.4), we see no way to prevent systems running code and processing data from revealing the code and data.

Confidentiality with respect to intermediaries is not a requirement any of the frameworks but ours. However, by dispensing with intermediaries entirely, Foster, et.al. could easily achieve confidentiality with respect to intermediaries; however, their stated requirements explicitly dispensed with the need for confidentiality and their implementation uses SSL for authentication but not for transport.

9. **Privacy**: *Some users or resources may require that the system not reveal their identity to other components processing the workflow.*

   None of these frameworks have attempted to address privacy issues.

10. **Trust relationships**: *Trust relationships must be explicit to support delegation.*

    In all of the frameworks but ours, trust relationships are either explicit or implicit (if a process is assumed to be trusted, it is assumed that it will only give data and credentials to other trusted processes). In our framework, neither explicit nor implicit trust is required.

11. **Integrity**: *Integrity of data and code "must be beyond reproach".*

    Ours is the only framework explicitly providing integrity assurances other than via point-to-point communications. Foster, et.al., provide integrity assurances for communications between user proxies and resource proxies because point-to-point communication is required. However, integrity checking is not explicitly performed other than during communications.

12. **Integrity of delegated rights**: *In addition to integrity of data and code, rights associated with delegated credentials must not be alterable.*

    Each of the systems uses digital signatures to make delegated credentials unalterable.

13. **Non-repudiation**: *Users must be able to determine that a resource agreed to run, or ran, their jobs. Users and resources must be able to determine their rights and responsibilities with respect to workflow transactions.*

    Ours is the only system where this information is collected, signed, and stored. Receipts in particular are used to link and record information about the specific results of a computation and the host on which it was computed. Since this information is recorded and stored with the results, it can be verified after the fact.

    In addition, the signatures on both job and machine ClassAds in our framework provide a basis for non-repudiation regarding responsibilities that form the basis of a match. Finally, limitations on the use of credentials and tasks is explicitly included in the delegation chain and available for resources to check.

14. **Freshness**: *System components must be able to determine credential freshness, especially with respect to authentication and authorization.*

    Each of the systems requires loosely synchronized clocks and include time stamps in delegated credentials limiting their validity periods.

15. **Task-specific credentials**: *Users should be able to link delegated credentials with a particular task so that the credentials may not be used for other tasks.*

    Our framework introduces task-specific credentials. Ours is the only system that links delegated credentials with a specific task and provides a mechanism for enforcing that link.

16. **Attribution and reproducibility**: *When the validity of some workflow results are challenged (i.e. because a host or site are found to have been compromised), it may be necessary to recalculate a subset of the results from the same inputs and verify that the same results were produced.*

Our framework explicitly records information about not only where a calculation was performed but can match checksums for input and output data. This could be particularly valuable in a situation where a complex workflow included tasks that are questionably valid, because it could limit the amount of recalculation required.

17. **Minimize trust**: *The number of processes that users and resources must trust should be kept to a minimum.*

    The goal of our framework is to limit the scope of processes that must be trusted.

Chapter 9

Related Work

The framework described in this thesis draws on several sources: it extends Condor and GSI and makes use of cryptographic primitives. Some of these sources are described in detail in Chapter 3. While the emphasis of that chapter is on the background necessary to understand our work, this chapter directly relates and contrasts our work to other work in the recent and current literature.

9.1 Proxy-based Delegation

Proxy-based authorization, including restricted proxies, was first introduced by Gasser and McDermott [43] and extended by Neuman [72]. Gasser and McDermott described a system based on public key cryptography that could be used to form delegation chains. This work is discussed below as well as in Chapter 8.

9.1.1 Restricted Delegation

When a delegation occurs, in the simplest case, the recipient of the delegation can *impersonate* the delegator; in other words, any action that the issuing certificate can perform can also be performed by the issued certificate. Many of the proxy-based authorization systems include mechanisms that could be used to further restrict the usage of the delegated credentials.

- Gasser and McDermott mention that their system for using certificates to perform delegation could include features that would permit restricted delegation but argue

that their roles system could be used to achieve the same goals. They do not specify a policy language for restricted delegation

- Neuman introduced the use of proxies as capabilities, permitting the use of hybrids of capabilities and access control lists, described the use of restricted delegation for authorization, and emphasized the use of proxies for accounting and audit in addition to authorization. Although Neuman's description of proxies includes support for restricted delegation and a scheme for proxies based on credentials using public-key encryption, this scheme is not as flexible or powerful as one based on certificates. Neuman's proposed policy language was a collection of typed fields, each field corresponding to a different restriction.

- The RFC covering proxy certificates [103], is based on work introduced in GSI [39] and extended to include support for delegation chains [104]. The proxy certificate standard includes a field for restricting the usage of the certificate according to a policy, leaving the policy language unspecified. Our framework builds on this foundation by specifying a policy expression language and interpretation.

9.2 Attribute-Based Authorization

A number of systems use attribute-based authorization [15, 105, 106, 8, 29, 102, 16, 32, 74] to determine what rights a user has in a Grid environment. These systems assume that the assertion of the user that they have those rights is insufficient and that they must provide evidence (or the resource must obtain evidence) that their attempts to access a resource are authorized by a *Virtual Organization* (VO), a third party trusted by the resource. These systems (in particular, CAS [75, 74, 26], VOMS [8, 9], and GridShib [15]) are complimentary to our approach, because our approach allows the user to specify how the rights granted to delegatees through these mechanisms should be limited.

Two versions of CAS are described in the literature. In the first version, resources trust VOs to set policy for their users; users obtain proxy certificates issued by a VO describing

their rights and use these as capabilities to obtain access to resources [75]. In the second version, the "authorization assertion" information is embedded in the user's proxy certificate in a way similar to how we create a task-specific proxy certificate. Instead of using the policy field in the proxy certificate, CAS uses a non-critical extension, permitting handling of the certificate chain by components not aware of the CAS extensions [74]. The authorization assertion itself is signed by the CAS server acting on behalf of the VO. The authorization assertion lists the rights granted to the user by the VO and is signed. The authors write that the assertion is in a proprietary format but that the authors are experimenting with SAML.

The purpose of CAS is to provide the local site administrators for each VO a way to express policy about group membership and access control for resources. In the original CAS implementation, capabilities in CAS are issued to "holder," who impersonates the user. In the second implementation, capabilities are bound to the proxy delegation chain, but not to a particular task. In the CAS model, the user's credential is enhanced rather than restricted by the capability issued to them. Although the mechanism permits it, it is not clear that the implementation could be used to limit or restrict delegated credentials – the user does not appear to have a way of specifying limits on the capability they request.

Similarly, the attribute-based authorization system Virtual Organization Membership Service (VOMS) permits users who authenticate to a local authority (a VOMS server) to obtain a (signed) statement detailing their group membership for authorization purposes. Their delegated credentials will then contain information that may factor in authorization decisions as their jobs are executed and access resources. This information usually takes the form of group membership, including any temporal roles than may expand access. This is very similar to CAS but there are some differences. In CAS, the attributes describe rights and are in a proprietary format but the VOMS server issues a statement of group membership in the form of an Attribute Certificate (AC).

Attribute-based authorization more recently has turned toward adopting the federation model as implemented in GridShib, based on GSI and Shibboleth [15, 88]. This is similar to previous techniques (CAS and VOMS) in that attributes are included in the proxy delegation

chain, and differs in the ways standards are used and interoperability is achieved. VOMS in particular uses a proprietary protocol, server, and assertion format, while GridShib uses SAML for the assertion format and Shibboleth services and protocols to transfer group membership information.

9.2.1 On-Demand and Dynamic Delegation

On-demand restricted delegation and dynamic delegation describe systems that delegate rights to perform actions, which are either not known or not specified when a task is submitted [5, 4, 95]. Ahsant, et. al., describe a system in which tasks perform "call-backs" to delegation services to obtain required credentials [4]. This system has the advantage that since the necessary credentials are obtained at runtime rather than at submission time, the submitter has the opportunity to disable access once granted. In contrast, in the framework described in this thesis, communication between the WN and another service are not required to obtain restricted credentials. While it might be argued that this presents a security risk because the submitter does not have the explicit ability to revoke credentials (there's no equivalent to certificate revocation lists for delegation chaining), the framework includes mechanisms that make revocation unnecessary. Framework task-specific credentials are already assumed to be in untrusted hands; framework mechanisms handle misuse without revocation.

The UNICORE model [95] introduces the idea that in an environment where user's signature is required to run a job as that user; a server's signature may be substituted when explicitly permitted by the recipient. This work is similar to ours in that it describes a method for digitally signing job descriptions in order to restrict delegation; however, there is no support for expressing policy in the job description or associating job descriptions with delegation chains.

The Fuzzy Trust and Delegation Model (FTDM) is a system in which a fuzzy inference process is used to compute whether delegated access should be permitted [45].

Jiang, et. al. [53, 52] propose a trust and reputation system for restricted delegation using attribute certificates. In this system, every time a delegation step is performed, an attribute certificate is generated which explicitly lists the delegated rights, the holder, and the issuer. This system differs from ours in that here, restricted delegations occur between communicating parties, rather than end-to-end; the delegation restrictions are not bound to either the task description or the delegation chain. The emphasis of this paper is on calculating trust and reputation metrics.

9.3 Mobile Agent Security

Tasks that move within a distributed batch computing system are similar to *mobile agents*.Doug Thain argued persuasively (in the context of Condor) that tasks within distributed batch computing systems should be represented by *agents* [99] but this is a different usage of the term agent. In Thain's work, the agents are processes that provide services to jobs in a distributed batch computing system. In the literature of mobile agents, agents are processes that clone themselves and move around in a distributed system.

Although mobile agents are similar to jobs within a distributed batch computing system, the security requirements for mobile agent systems differ significantly from those in distributed batch computing, primarily because the agents differ in their capabilities and responsiblities [61, 70]. In a mobile agent system, the agents actually execute on all the "intermediaries" within the system, while in a distributed batch computing system the jobs only execute on a specific set of worker nodes. Once jobs complete, they are not expected to move or run again (obviously Condor's facilities for checkpointing and migration are an exception to this statement but the difference here is that with mobile agents, the goal of the agents is to migrate whereas in Condor it is a side effect).

In a masters thesis and subsequent paper [77, 78], Subhashini Raghunathan describes the design a mobile agent security system based on GSI that has some similarities to the framework described in this thesis, but differs as a result of the differences between mobile

agents and distributed batch computing jobs. Notably, agent code and data is cryptographically checksummed and included in the proxy delegation chain, and a mechanism exists for preventing execution on untrusted hosts that is similar to service-specific proxy certificates. Because of the differences between mobile agents and jobs, there are several mechanisms described in this thesis that are not present in Raghunathan's work, including references to external data in signed ClassAds, action authorization expressions leveraging the ClassAd language, and confidentiality mechanisms.

9.4 Trusted Computing

Trusted computing (TC) [65, 85, 76, 47, 58, 20, 40] provides one approach to securing the execution host where jobs run. Our framework assumes that execution hosts are trusted, but provides no mechanisms for providing trust guarantees. TC can provide assurances that a particular host or service has not been tampered with, even by users with physical access to the host. In particular, the Daonity project addresses the use of TC in Grid computing [64, 63].

9.5 Audit and Provenance

There are similarities between the security requirements for jobs in a distributed batch computing system and those described in the literature for provenance [93]. Provenance issues have been explored before in the context of Condor [80, 81]. However, this work does not address the issues related to secure provenance [48, 50, 49].

One approach to secure provenance within Grid environments is to abstract all Grid operations as a Service Oriented Architecture (SOA): components take inputs and produce outputs, and workflows compose these components [98]. In this approach, a *p-assertion* is defined as a specific piece of information documenting the action of one component. A *provenance store* is a repository holding p-assertions that can perform queries on data provenance and return sets of p-assertions. There are several notable contributions here: access control for process documentation, a trustworthiness or reputation metric for producers of

p-assertions. One aspect that is similar to our approach is that services can digitally sign the results of the work they perform, providing a link between the signer and the signed data. This is similar to our *receipts*; possibly they could be used as p-assertions, or extended to include p-assertion data.

9.6 Performance

With regard to performance, Shirasuna, et. al. compare message-based security with session-based security and conclude that the latter is faster [92]. Although this apparently contradicts our performance results that describe a workload where message-based security (in the form of signed ClassAds) can improve performance, in fact these results are consistent with ours. Condor already relies extensively on session-based security. The performance improvements resulting from message-based security described in Chapter 7 apply to a particular component within the distributed infrastructure, a heavily loaded `condor_collector` daemon interacting with many different execute machines, and we do not suggest that session-based security should be scrapped in Condor, merely that it should be adopted in situations like this where it can improve performance. Furthermore, our system does not suffer from the performance overheads associated with XML parsing that dominate their measurements.

Chapter 10

Conclusions and Future Work

Basic well-understood security mechanisms such as encryption and integrity checks are not present end-to-end in many distributed batch computing systems. This could lead to attacks which would bring the validity of many expensive and time-consuming scientific calculations into question. Furthermore, existing systems don't offer security mechanisms that allow the user submitting jobs to specify constraints on how their tasks, data and credentials may be used.

We have described a framework that addresses many of the risks present in these systems and permits users to specify policy for their jobs and credentials; we have implemented this framework and analyzed its performance, and analyzed it in comparison with other systems providing security for distributed batch computing environments.

This framework provides the following properties:

Task, input, and output integrity and confidentiality: The integrity of task input and output data is protected from untrustworthy schedulers in the task pathway through explicit checks on task integrity performed at the endpoints: the WN checks the task ClassAd it receives from the schedulers against the signed ClassAd contained in the delegation chain, including any forwarded data or executables, and the submitter checks the signature on the receipt when the task is complete. A secure key storage service permits end-to-end confidentiality.

Integrity and confidentiality of data unrelated to the task: Because of the additional information included in the proxy delegation chain, submitters can explicitly specify

policy that limits the access the infrastructure has to data unrelated to the task. In addition, the framework limits access to related data to specific infrastructure components in the task pathway when they are acting on behalf of the task.

Protection of the operating environment of the worker node: The operating environment of the WN is protected because it can authenticate the source of tasks it receives and perform explicit authorization based on the identity of the submitter. The worker node is able to verify the integrity of tasks it receives, and use the information about task origin to collect much stronger audit information than is currently possible.

User controllable fine grained authorization: In the system we describe, submitters have control over the restrictions placed on the usage of delegated credentials. Delegated credentials can be limited to the set of actions needed to perform a particular task.

Credentials tied to task: The signed ClassAd approach allows users to tightly bind tasks to credentials when they submit a task as well as to specify restrictions based on task properties. When a request is made on behalf of a user, the destination of the request can verify that the request is consistent with the task description and record information about the task in relevant audit logs and provenance mechanisms. For example, when a task is running on an execute machine, it may use the accompanying credentials to read or write files on resources that the user has access to. When these resource requests are made the signed ClassAd specifying the task must accompany the proxy certificate chain used for authentication. The resource can record the task information for audit purposes, and use this information for authorization decisions.

Policy expressions limit credential usage based on participants: Policy in the signed ClassAd environment is expressed in the ClassAd language. Enforcement of task and result integrity occurs at the trusted endpoints: the task submitter and WN. Enforcement of access control for resources is enforced by the resource, given information provided by the submitting user and the WN. The combination of authorization expressions and service-specific proxy certificates allows us to ensure confidentiality by limiting access to authorized holders of the credentials. This is used to implement end-to-end confidentiality.

In the rest of this chapter, we summarize the framework and its analysis, then discuss general lessons learned while designing and implementing the framework, and finally outline directions for future research.

10.1 Summary

To summarize the work presented in this thesis, we summarize the the framework of security mechanisms we have designed and implemented, then discuss our analysis of how this framework meets the performance and security requirements present in many distributed batch computing operating environments.

10.1.1 Framework

Framework components include:

- Signed ClassAds
- Task-specific proxy certificates
- Action authorization expressions
- Service-specific proxy certificates
- Transparent credential transformation
- Key storage service

10.1.2 Performance Analysis

We have analyzed the performance of the framework and found that the overhead associated with employing the framework is negligible given reasonable assumptions about job characteristics. Only when framework mechanisms are provided for large jobs with short run times do they make an observable impact on total job run-time.

10.1.3 Security Analysis

Finally, we have analyzed the security features of the framework and compared them to other implementations. The key requirements addressed by this system that are not addressed by other implementations include:

- End-to-end confidentiality and integrity.
- User specified policy and restricted delegation.
- Task and service specific credentials.
- Reduced scope of required trust assumptions.
- Non-repudiation, attribution and reproducibility of task inputs and outputs.

10.2 Lessons Learned

In this section, we briefly discuss lessons learned while working on this thesis.

Condor itself is end-to-end, making it particularly well suited for this type of research. Because distributed batch computing systems perform a wide variety of functions, including data management, job control, authentication, authorization, etc., they often include many components. Condor in particular includes more than 100 tools and daemons in a standard installation, and Condor interoperates with several other systems including Globus Toolkit, NorduGrid, Unicore, PBS, LSF and Amazon EC2. However, when using Condor it is quite common to use tools provided with Condor for all of these functions so that your jobs and data are handled by Condor exclusively.

Condor's internal libraries include ClassAds, described in Chapter 3, DaemonCore, used to manage the event loop for daemon processes and by tools and daemons to communicate, and CEDAR, used to perform security negotiation for authentication and authorization, for message integrity and encryption and wrapping network communications. These libraries are used pervasively within the Condor code base, so features implemented in them can easily be used within any of the many tools and daemons.

Without such a tightly coordinated centrally organized code base, it would have been much more difficult to implement many of the framework mechanisms described in this thesis.

Security features in Grid computing have not been driven by active attackers. In hindsight, many of the security mechanisms described in this thesis are sensible adaptations of existing state of the art for simpler scenarios (such as client-server communications) to the more complex distributed batch computing environment. So why is so much of the research and development in Grid computing focused on authentication and authorization at the expense of basic mechanisms like end-to-end integrity checks and secure audit mechanisms? We are of the opinion that this is because much of the feature development has been driven by integration and interoperation goals rather than actual attacks revealing security flaws.

This may change as commoditization of computing infrastructure occurs (i.e. "Cloud computing.") Attackers may find that once they become familiar with the complexities of distributed batch computing software, in many installations they find an open window next to a locked door. We applaud efforts such as the work of Kupsch and Miller [60, 59] to analyze the security of these systems with a vulnerability assessment approach and look forward to more effort in this area.

Retrofitting security is a necessary activity. One of the standard design principles for security is to, "include security in design from the start." [23] While this is certainly a good idea, there are many systems, such as Condor, that are being used in ways their designers never intended, ways that result in different security concerns than were considered when they were designed. As a result, it is necessary to adapt these systems to include security mechanisms that will satisfy new security requirements: *retrofitting* security. When new business models and innovations in technology drive change in complex software, retrofitting for security is almost certain to be required.

However, this retrofitting can not take the form of simply adding security features without consideration of the design and usage of the system as a whole – instead the mechanisms that are added must anticipate and defend against weak points in the system as a whole.

Adding security to a system can make it useful in a wider range of contexts. Tying together the themes of this section, we have found that it is possible to retrofit security features into Condor because it has a tightly coordinated centrally organized code base. We have tried to approach this effort as a redesign rather than simply adding security features, and by anticipating the ways that potential attackers might find weaknesses in the whole system.

We believe that by adding our framework of mechanisms, we have extended the range of contexts in which Condor can be a useful tool. Specifically, by shifting the burden of trust away from job intermediaries, we hope to facilitate sharing between administrative domains that could not previously interoperate.

In science, business, among information technology professionals as well as among the general public, security is often viewed as a barrier to "getting work done" rather than as a facilitator.[1] In fact this view is held by many computer scientists including researchers and developers specifically focused on distributed batch computing. Upon reflection, it seems quite obvious that the opposite is true: without appropriately designed and implemented security infrastructure, security requirements would prevent much of the internet from being useful.

10.3 Future Work

In this section, we outline various directions for additional research related to the security framework described in this thesis. We discuss first connections to secure provenance, then describe ways that the authentication and authorization infrastructure in Condor could be

[1]For example, in the popular comic strip Dilbert, by Scott Adams, the security administrator character Mordac's title is "Preventer of Information Services." As he says in one strip, "Security is more important than usability. In a perfect world, no one would be able to use anything."

extended to be purely message-based. Finally we discuss ways that this work could be extended to work in additional environments, both within Condor as well as beyond Condor.

10.3.1 Message based authentication

Many messages exchanged by processes in Condor take the form of ClassAds. In particular, the `condor_collector` process receives ClassAds from all processes. Our work on performance described in Chapter 7 shows that message based authentication can provide performance improvements in the common case where many `condor_startd` processes are sending updates to `condor_collector` at the same time. There may be other components within the Condor system that could also take advantage of this approach.

In addition to providing performance benefits in situations where many processes are communicating simultaneously, this approach could also provide benefits by making the transport mechanism more flexible. For example, many organizations permit access to services via HTTP but not other protocols; message based authentication could be used to encapsulate regular Condor communications within HTTP.

In the present implementation, this would only be appropriate for communications where integrity is required but not confidentiality; but security analysis may show that there are situations where this is acceptable (as in the case of `condor_startd` processes) or additional mechanisms may be employed to permit confidentiality. In order to implement this, the basic mechanisms that are used to perform communications within Condor (the CEDAR libraries) would need to be restructured or expanded.

10.3.2 Integration with Glide-Ins

One interesting area of research for security within Condor is in the case of Glide-Ins [100, 21]. Glide-Ins are a technique for creating dynamic Condor pools using arbitrary scheduling technology by running Condor daemons on top of other systems. When a Condor Glide-In is deployed as a job, it runs on some resource, and then connects by prior arrangement to an external Condor pool advertising itself as a resource capable of running Condor jobs.

Extending our work to this scenario would not be trivial; in the framework described in this thesis (specifically service-specific proxy certificates), we assume that the Condor daemons running on the remote resource have access to the host key (implying that they run as root). In order to grant this level of access, a trusted process would need to manage both the job and the glide-in. One candidate for this would be gLExec [91].

10.3.3 Secure provenance

Future efforts to develop systems providing secure provenance could take advantage of the information describing jobs and input and output data contained in signed ClassAds, including cryptographic checksums, as a basis for records describing the process of calculating a particular result.

Provenance records have been used for many years in archives, art, and archaeology to list the ownership history of an item and the actions performed on it. More recently, the technique has been applied to modifications of digital documents, including work scheduled and managed by Condor [81]. Secure provenance extends this trend by making provenance records robust against attackers who have access to a provenance chain and want to alter it inappropriately [48, 50, 49].

The process of signing a ClassAd for submission, or of signing a receipt for return, involves a principal signing a hash of a document, as does the process for creating *signature-based checksums* in Secure Provenance [49]. This connection could be explored more fully.

LIST OF REFERENCES

[1] Data Encryption Standard. FIPS pub 46-3, U.S. National Bureau of Standards.

[2] Security Support Provider Interface. Technical report, Microsoft Corp.

[3] A. Abdul-Rahman and S. Hailes. Supporting trust in virtual communities. In *Proceedings of 33rd Hawaii International Conference on System Sciences*, 2000.

[4] Mehran Ahsant, Jim Basney, and Lennart Johnsson. Dynamic, Context-Aware, Least-Privilege Grid Delegation. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing (Grid2007)*, September 2007.

[5] Mehran Ahsant, Jim Basney, Olle Mulmo, Adam J. Lee, and Lennart Johnsson. Toward An On-demand Restricted Delegation Mechanism for Grids. In *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, Barcelona, September 2006.

[6] Ian D. Alderman. Social, legal, political and economic questions relevant for research in the deployment of scalable cyber-security in large-scale networks. *Workshop on Scalable Cyber-Security Challenges in Large-Scale Networks: Deployment Obstacles*, March 2003.

[7] Ian D. Alderman and Miron Livny. Task-specific restricted delegation. In *HPDC '07: Proceedings of the 16th international symposium on High performance distributed computing*, 2007.

[8] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lőrentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Gener. Comput. Syst.*, 21(4):549–558, 2005.

[9] R. Alfieri, R. Cecchini, V. Ciaschini, Á. Frohner, A. Gianoli, K. Lőrentey, and F. Spataro. VOMS, an authorization system for virtual organizations. In F. Fernández Rivera et al., editor, *Across Grids 2003*, number 2970 in LNCS. Springer-Verlag, 2004.

[10] Ross Anderson. Why Information Security is Hard – An Economic Perspective. In *Proceedings of the 17th Computer Security Applications Conference*, pages 358–365, 2001.

[11] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

[12] Anil L. Pereira, Vineela Muppavarapu, and Soon M. Chung. Role-Based Access Control for Grid Database Services Using the Community Authorization Service. *Ieee Transactions On Dependable And Secure Computing*, 3(2):156–166, 2006.

[13] Farag Azzedin and Muthucumaru Maheswaran. Integrating Trust into Grid Resource Management Systems. *2002 International Conference on Parallel Processing*, pages 47–54, August 2002.

[14] Farag Azzedin and Muthucumaru Maheswaran. A trust brokering system and its application to resource management in public-resource grids. In *Proceedings of International Parallel and Distributed Processing Symposium*, Santa Fe, NM, April 2004.

[15] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Monte Goode, and Kate Keahey. Identity Federation and Attribute-based Authorization through the Globus Toolkit, Shibboleth, GridShib, and MyProxy. In *5th Annual PKI R&D Workshop*, 2006.

[16] J. Basney, S.S. Chetan, F. Qin, S. Song, X. Tu, and M. Humphrey. An OGSI CredentialManager Service. In *Proceedings of the UK Workshop on Grid Security Experiences*, Oxford, July 2004.

[17] Jim Basney, Terry Fleury, and Von Welch. Federated Login to TeraGrid. In *9th Symposium on Identity and Trust on the Internet (IDtrust 2010)*, Gaithersburg, MD, April 2010.

[18] Jim Basney, Marty Humphrey, and Von Welch. The MyProxy Online Credential Repository. *Software: Practice and Experience*, 35(9):801–816, July 2005.

[19] Jim Basney, William Yurcik, Rafael Bonilla, and Adam Slagell. The Credential Wallet: A Classification of Credential Repositories Highlighting MyProxy. In *31st Research Conference on Communication, Information and Internet Policy (TPRC 2003)*, Arlington, VA, September 2003.

[20] Eberhard Becker, Dirk Günnewig Willms Buhse, and Niels Rump, editors. *Digital Rights Management: Technological, Economic, Legal and Political Aspects*. Springer-Verlag, 2003.

[21] Stefano Belforte, Matthew Normal, Subir Sarkar, Igor Sfiligoi, Douglas Thain, and Frank Wuerthwein. Using condor glide-ins and parrot to move from dedicated resources to the grid. In *Lecture Notes in Informatics*, volume 81, pages 285–292. March 2006.

[22] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology - Crypto '93 Proceedings, LNCS Vol. 773*. Springer-Verlag, 1994.

[23] T. V. Benzel, C. E. Irvine, T. E. Levin, G. Bhaskara, T. D. Nguyen, and P. C. Clark. Design principles for security. Technical Report NPS-CS-05-010, Naval Postgraduate School, September 2005.

[24] Philippa J. Broadfoot and Gavin Lowe. Architectures for Secure Delegation within Grids. Technical Report PRG-RR-03-19, Oxford University Computing Laborotory, August 2003.

[25] Philippa J. Broadfoot and Andrew P. Martin. A Critical Survey of Grid Security Requirements and Technologies. Technical Report PRG-RR-03-15, Oxford University Computing Laborotory, September 2003.

[26] S. Cannon, S. Chan, D. Olson, C. Tull, V. Welch, and L. Pearlman. Using CAS to Manage Role-Based VO Sub-Groups. In *Proceedings of Computing in High Energy Physics '03 (CHEP '03)*, 2003.

[27] CCITT. The directory authentification framework. Draft Recommendation X.509, 1988.

[28] David Chadwick. Authorisation in Grid Computing. *Information Security Technical Report*, 10(1):33–40, 2005.

[29] D.W. Chadwick, A. Otenko, and E. Ball. Role-Based Access Controls Using X.509 Attribute Certificates. *IEEE Internet Computing*, pages 62–69, March 2003.

[30] Joan Daemen and Vincent Rijmen. *The design of Rijndael : AES–the Advanced Encryption Standard*. Springer, 2002.

[31] E. Deelman, G. Singh, M.P. Atkinson, A. Chervenak, N.P. Chue Hong, C. Kesselman, S. Patil, L. Pearlman, and M. Su. Grid-Based Metadata Services. In *Proceedings of 16th International Conference on Scientific and Statistical Database Management (SSDBM04)*, June 2004.

[32] S. De Capitani di Vimercati and P. Samarati. New Directions in Access Control. In *Cyberspace Security and Defense: Research Issues*, pages 279–296. Kluwer Academic Publishers, May 2005.

[33] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), October 1997.

[34] Whitfield Diffie and Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[35] Donald Eastlake and Joseph Reagle. XML Encryption Syntax and Processing. W3C Recommendation, 2002.

[36] Donald Eastlake, Joseph Reagle, and David Solo. XML-Signature Syntax and Processing. W3C Recommendation, 2002.

[37] C. Ellison and B. Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[38] Charles Patrick Ewing. Introduction to threat assessment. *Behavioral Sciences and the Law*, 17, 1999.

[39] Ian Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[40] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *SOSP*, pages 193–206. ACM Press, 2003.

[41] G. Garzoglio, I. Alderman, M. Altunay, R. Ananthakrishnan, J. Bester, K. Chadwick, V. Ciaschini, Y. Demchenko, A. Ferraro, A. Forti, D. Groep, T. D. Hesselroth, J. Hover, O. Koeroo, C. La Joie, T. Levshina, Z. Miller, J. Packard, H. Sagehaug, I. Sfiligoi, N. Sharma, F. Siebenlist, S. Timm, V. Venturi, and J. Weigand. XACML Profile and Implementation for Authorization Interoperability between OSG and EGEE. In *17th International Conference on Computing in High Energy and Nuclear Physics (CHEP 2009)*, May 2009.

[42] Gabriele Garzoglio, Ian Alderman, Mine Altunay, Rachana Ananthakrishnan, Joe Bester, Keith Chadwick, Vincenzo Ciaschini, Yuri Demchenko, Andrea Ferraro, Alberto Forti, David Groep, Ted Hesselroth, John Hover, Oscar Koeroo, Chad La Joie, Tanya Levshina, Zach Miller, Jay Packard, Håkon Sagehaug, Valery Sergeev, Igor Sfiligoi, Neha Sharma, Frank Siebenlist, Valerio Venturi, and John Weigand. Definition and implementation of a saml-xacml profile for authorization interoperability across grid middleware in osg and egee. *Journal of Grid Computing*, 7(3), September 2009.

[43] Morrie Gasser and Ellen McDermott. An Architecture for Practical Delegation in a Distributed System. *IEEE Symposium on Research in Security and Privacy*, pages 20–30, May 1990.

[44] Dan Geer. Risk Management is Where the Money Is. Presented at The Digital Commerce Society of Boston, MA, November 1998. Widely reprinted, available at http://catless.ncl.ac.uk/Risks/20.06.html.

[45] G. Geethakumari, Atul Negi, and V. N. Sastry. Dynamic Delegation Approach for Access Control in Grids. In *E-SCIENCE '05: Proceedings of the First International Conference on e-Science and Grid Computing*, pages 387–394. IEEE Computer Society, 2005.

[46] P. Golle and I. Mironov. Uncheatable distributed computations. In *Proceedings of the RSA Conference 2001, Cryptographer's Track*, Lecture Notes in Computer Science, pages 425–441, San Francisco, CA, 2001.

[47] Stuart Haber, Bill Horne, Joe Pato, Tomas Sander, and Robert Endre Tarjan. *If Piracy is the Problem, is DRM the Answer?*, pages 232–241. In Becker et al. [20], 2003.

[48] Ragib Hasan, Radu Sion, and Marianne Winslett. Introducing secure provenance: problems and challenges. In Valerie Henson, editor, *StorageSS*, pages 13–18. ACM, 2007.

[49] Ragib Hasan, Radu Sion, and Marianne Winslett. Preventing History Forgery with Secure Provenance. *ACM Transactions on Storage*, November 2009.

[50] Ragib Hasan, Radu Sion, and Marianne Winslett. The Case of the Fake Picasso: Preventing History Forgery with Secure Provenance. In *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST)*, 2009.

[51] Marty Humphrey, Mary Thompson, and Keith Jackson. Security for Grids. *Proceedings of the IEEE (Special Issue on Grid Computing)*, 93(3), March 2005.

[52] Wenbao Jiang, Song Hua, and Yiqi Dai. Realizing restricted delegation using attribute certificates in grid environments. In *SNPD*, pages 359–365. IEEE Computer Society, 2004.

[53] Wenbao Jiang, Chen Li, Shuang Hao, and Yiqi Dai. Using trust for restricted delegation in grid environments. In *Proceedings of the First Information Security Practice and Experience Conference (ISPEC 2005), LNCS 3439*, pages 293–301, 2005.

[54] Kate Keahey and Von Welch. Fine-Grain Authorization for Resource Management in the Grid Environment. In *Proceedings of Grid2002 Workshop*, 2002.

[55] Gene H. Kim and Eugene H. Spafford. Experiences with Tripwire: Using integrity checkers for intrusion detection. In *Proceedings of The Third Annual System Administration, Networking and Security Conference (SANS III)*, pages 89–101, Washington, DC, April 1994.

[56] L. M. Kohnfelder. Towards a practical public-key cryptosystem, May 1978.

[57] Olga Kornievskaia, Peter Honeyman, Bill Doster, and Kevin Coffman. Kerberized Credential Translation: A Solution to Web Access Control. In *Proceedings of the USENIX Security Symposium*, Washington, DC, 2001.

[58] Dirk Kuhlmann and Robert A. Gehring. *Trusted Platforms, DRM and Beyond*, pages 186–213. In Becker et al. [20], 2003.

[59] James A. Kupsch and Barton P. Miller. Manual vs. automated vulnerability assessment: A case study. In *First International Workshop on Managing Insider Security Threats (MIST 2009)*, West Lafayette, IN, June 2009.

[60] James A. Kupsch, Barton P. Miller, Eduardo César, and Elisa Heymann. First principles vulnerability assessment. Technical report, MIST Project, September 2009.

[61] Byoungcheon Lee, Heesun Kim, and Kwangjo Kim. Secure Mobile Agent using Strong Non-designated Proxy Signature. In *Proc. of ACISP2001, LNCS 2119*, pages 474–486. Springer-Verlag, 2001.

[62] Michael Litzkow, Miron Livny, and Matthew Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, June 1988.

[63] W. Mao, A. Martin, H. Jin, and H. Zhang. Innovations for Grid Security from Trusted Computing – Protocol Solutions to Sharing of Security Resource. In *14th International Workshop on Security Protocols*, Cambridge University, March 2006.

[64] W. Mao, F. Yan, and C. Chen. Daonity – Grid Security with Behaviour Conformity from Trusted Computing. In *1st ACM Workshop on Scalable Trusted Computing*, November 2006.

[65] John Marchesini, Sean W. Smith, Omen Wild, and Rich MacDonald. Experimenting with TCPA/TCG Hardware, Or: How I Learned to Stop Worrying and Love The Bear. Technical Report TR2003-476, Dartmouth College, Computer Science, Hanover, NH, December 2003.

[66] Ueli Maurer. New Approaches to Digital Evidence. *Proceedings of the IEEE*, 92(6):933–947, June 2004.

[67] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *The Handbook of Applied Cryptography*. CRC Press, October 1996.

[68] J. Meyers. Simple Authentication and Security Layer (SASL). RFC 2222 (Proposed Standard), October 1997.

[69] Barton P. Miller, Mihai Christodorescu, Robert Iverson, Tevfik Kosar, Alexander Mirgorodskii, and Florentina Popovici. Playing Inside the Black Box: Using Dynamic Instrumentation to Create Security Holes. *Parallel Processing Letters*, 11(2/3), June/September 2001.

[70] Srilekha S. Mudumbai, William Johnston, and Abdelilah Essiari. Anchor Toolkit - a secure mobile agent system. Technical Report 2594j56c, Lawrence Berkeley National Laboratory, http://www.escholarship.org/uc/item/2594j56c, 2008.

[71] Lik Mui, Mojdeh Mohashemi, and Ari Halberstadt. A computational model of trust and reputation. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, 2002.

[72] B. Clifford Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, Pittsburgh, May 1993.

[73] Sangmi Lee Pallickara and Beth Plale. Enabling End-to-End Trustworthiness in Data-Oriented Scientific Computing. In *Proceedings of the Workshop on Web Services-based Grid Applications (WGSA '06) in association with the International Conference on Parallel Processing (ICPP-06)*, August 2006.

[74] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The Community Authorization Service: Status and Future. In *Proceedings of Computing in High Energy Physics '03 (CHEP '03)*, 2003.

[75] L. Pearlman, Von Welch, Ian Foster, Carl Kesselman, and Steven Tuecke. A community authorization service for group collaboration. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002.

[76] Siani Pearson, editor. *Trusted Computing Platforms: TCPA Technology in Context.* Prentice Hall PTR, 2003.

[77] Subhashini Raghunathan. A Security Model for Mobile Agents using X.509 Proxy Certificates. Master's thesis, University of North Texas, December 2002.

[78] Subhashini Raghunathan, Armin R. Mikler, and Cliff Cozzolino. Secure agent computation: X.509 proxy certificates in a multi-lingual agent framework. *J. Syst. Softw.*, 75(1-2):125–137, 2005.

[79] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7)*, Chicago, IL, July 1998.

[80] Christine F. Reilly and Jeffrey F. Naughton. Exploring Provenance in a Distributed Job Execution System. In *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW 2006)*, May 2006.

[81] Christine F. Reilly and Jeffrey F. Naughton. Transparently Gathering Provenance with Provenance Aware Condor. In *Proceedings of the First Workshop on the Theory and Practice of Provenance (TAPP 2009)*, February 2009.

[82] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Proposed Standard), April 1992.

[83] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[84] Paul Ruth, Dongyan Xu, Bharat Bhargava, and Fred Regnier. e-Notebook Middleware for Accountability and Reputation Based Trust in Distributed Data Sharing Communities. In *Proc. of the Second International Conference on Trust Management (iTrust 2004)*, Oxford, UK, March 2004. as LNCS Vol. 2995, Springer Verlag.

[85] David Safford, Jeff Kravitz, and Leendert van Doorn. Take Control of TCPA. Linux Journal, WWW, 2003.

[86] Jerome H. Saltzer, David P. Reed, and David D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.

[87] Jerome H. Saltzer and Mike D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), September 1975.

[88] Tom Scavo and Von Welch. A grid authorization model for science gateways. In *International Workshop on Grid Computing Environments*, 2007.

[89] B. Schneier. Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings*, pages 191–204, December 1993.

[90] Bruce Schneier. Attack Trees. *Dr. Dobb's Journal*, 24(12):21–29, December 1999.

[91] I. Sfiligoi, G. Quinn, C. Green, and G. Thain. Pilot job accounting and auditing in open science grid. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 112–117, Washington, DC, USA, 2008. IEEE Computer Society.

[92] Satoshi Shirasuna, Aleksander Slominski, Liang Fang, and Dennis Gannon. Performance Comparison of Security Mechanisms for Grid Services. In *In 5th IEEE/ACM International Workshop on Grid Computing*, pages 360–364. IEEE Computer Society Press, 2004.

[93] Yogesh L. Simmhan, Beth Plale, and Dennis Gannon. A Survey of Data Provenance Techniques. Technical Report 612, Computer Science Department, Indiana University, 2005.

[94] G. Singh, S. Bharathi, A. Chervenak, E. Deelman, C. Kesselman, M. Mahohar, S. Pail, and L. Pearlman. A Metadata Catalog Service for Data Intensive Applications. In *Proceedings of Supercomputing 2003 (SC2003)*, November 2003.

[95] Dave Snelling, Sven van den Berge, and Vivian Li. Explicit Trust Delegation: Security for Dynamic Grids. *Fujitsu Scientific & Technical Journal - Special Issue on Grid Computing*, 40(2), December 2004.

[96] J.G. Steiner, C. Neumann, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Proceedings of USENIX*, pages 191–200, 1988.

[97] D. Szajda, B. Lawson, and J. Owen. Hardening functions for large-scale distributed computations. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, pages 216–224, Berkeley, CA, May 2003.

[98] Victor Tan, Paul Groth, Simon Miles, Sheng Jiang, Steve Munroe, Sofia Tsasakou, and Luc Moreau. Security Issues in a SOA-based Provenance System. In *Proceedings of the International Provenance and Annotation Workshop 2006 (IPAW 2006)*, May 2006.

[99] Douglas Thain. *Coordinating Access to Computation and Data in Distributed Systems*. PhD thesis, University of Wisconsin, August, 2004.

[100] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid. In Fran Berman, Geoffrey Fox, and Anthony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., April 2003.

[101] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency - Practice and Experience*, 17(2-4):323–356, 2005.

[102] Mary R. Thompson, Abdelilah Essiari, and Srilekha Mudumbai. Certificate-based authorization policy in a PKI environment. *ACM Trans. Inf. Syst. Secur.*, 6(4):566–588, 2003.

[103] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard), June 2004.

[104] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, and F. Siebenlist. X. 509 proxy certificates for dynamic delegation. In *3rd annual PKI R&D workshop*, 2004.

[105] Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus Integration to Facilitate Grid Collaboration. In *4th Annual PKI R&D Workshop*, April 2005.

[106] Von Welch, Ian Foster, Tom Scavo, Frank Siebenlist, Charlie Catlett, Jill Gemmill, and Dane Skow. Scaling teragrid access: A testbed for identity management and attribute-based authorization. In *TeraGrid 2007*, 2007.