Authentication in peer-to-peer systems

By
Jonas Åslund

LiTH-ISY-EX-3153-2002

2002-05-16
Authentication in peer-to-peer systems

Examensarbete utfört i informationsteori
Vid Linköpings tekniska högskola
av
Jonas Åslund

Reg.nr: LiTH-ISY-EX-3153-2002

Handledare: Alf Bengtsson, FOI
Examinator: Viiveke Fåk, ISY

Linköping 2002-05-16
Sammanfattning
Title
Authentication in peer-to-peer systems
Författare
Author
Jonas Åslund

Abstract
In the environment of the 3:rd generation Internet based on peer-to-peer architecture, well-trusted methods must exist to establish a secure environment. One main issue is the possibility to verify that a node actually is who it claims to be (authentication). Establishment of authentication between nodes in a peer-to-peer environment where nodes are exchanging information directly with each other requires more planning than in a typical client-server environment where the authentication methods are server-based. The peer-to-peer applications described in this report use authentication methods based on central authorities as well as solutions without central authorities.

Lack of standards in the way peer-to-peer systems should communicate and apply security lead to a variety of “local” communication and security solutions. These local solutions make different applications incompatible with each other, meaning that a peer using one application will not be able to communicate and exchange information with other peers using some other application.

Nyckelord
Keyword
P2P, peer-to-peer, peer, node, authentication, public key, certificate, ticket
Abstract

In the environment of the 3:rd generation Internet, based on peer-to-peer architecture, well-trusted methods must exist to establish security. One main issue, in any system, is the possibility to verify that a node actually is who it claims to be (authentication). Establishment of authentication between nodes in a peer-to-peer environment requires a bit more planning than in the typical client-server environment to get a trusted and secure solution.

In a peer-to-peer system nodes (peers) are exchanging information directly with each other, instead of using some central stand-alone server. Two main architectures of peer-to-peer networks are available today, the pure model (without any central devices) and the hybrid model (with some central servers). The hybrid model provides, sometimes, a classic client-server authentication towards the central server, but not always. Hybrid, as well as pure peer-to-peer environments can use central authorities in the aspect of providing a trusting environment during the authentication process. Pure peer-to-peer systems will, in most cases, use central authorities to create initial peer identities and to distribute certificates and tickets, but their communication procedure is totally independent of any central unit.

The most used authentication methods in the peer-to-peer applications described in this report are some kind of identity verification based on public keys. This includes also the use of certificates, which uses the peers’ identity and public key. All certificates must be created and signed by a trusted central Certification Authority. Some of the applications and platforms also include support for tickets, provided by a Kerberos server, to authenticate other peers and to establish a secure connection.

JXTA, .NET, and Peer-to-Peer Trusted Libraries, are platforms (or frameworks) that provide good foundations for developers to build new secure peer-to-peer application upon. These platforms include different levels of security and authentication methods, to suit most developer’s requests. The evolution of these platforms will continue, along with the continuously adding of new functions and security features to their environments.
Lack of standards in the way peer-to-peer systems should communicate and apply security issues lead to a variety of “local” communication and security solutions. These local solutions make different applications incompatible with each other, which means that a peer, using one application, will not be able to communicate and exchange information with some other peer, using another application. Standardized communication methods would allow peers from different kinds of networks, and applications, to communicate and exchange information with each other.
# Content

1 Introduction .......................................................................................................................... 1  
1.1 Background....................................................................................................................... 1  
1.2 Motivation......................................................................................................................... 1  
1.3 Purpose ............................................................................................................................ 1  
1.4 About this document......................................................................................................... 2  

2 Peer-to-Peer: definitions and architecture ........................................................................ 5  
2.1 Definitions ....................................................................................................................... 6  
2.2 Architecture .................................................................................................................... 10  
  2.2.1 Hybrid peer-to-peer networks .................................................................................. 14  
  2.2.2 Pure peer-to-peer networks ....................................................................................... 15  
2.3 Summary ......................................................................................................................... 16  

3 Security: definitions........................................................................................................... 17  
3.1 Establishment of a trusted environment ......................................................................... 18  
3.2 Authentication.................................................................................................................. 19  
3.3 Authorization and access control ................................................................................. 21  
3.4 Data integrity .................................................................................................................. 24  

4 Network authentication methods ........................................................................................ 27  
4.1 Key distribution .............................................................................................................. 27  
  4.1.1 Private keys ............................................................................................................ 27  
  4.1.2 Public keys ............................................................................................................ 28  
4.2 Symmetric algorithms .................................................................................................... 29  
4.3 Asymmetric algorithms ................................................................................................. 31  

5 Peer-to-Peer authentication ............................................................................................... 33  
5.1 Napster ............................................................................................................................ 33  
5.2 Mangomind ..................................................................................................................... 34  
5.3 WebRiposte ..................................................................................................................... 34  
5.4 Groove ............................................................................................................................ 35  
5.5 Magi Enterprise .............................................................................................................. 36  
5.6 Avaki ............................................................................................................................... 37  
5.7 Globus ............................................................................................................................. 39  
5.8 Summary ......................................................................................................................... 40  

6 Secure P2P development platforms .................................................................................. 43  
6.1 JXTA ............................................................................................................................... 43  
6.2 .NET ............................................................................................................................... 44  
6.3 Peer-to-Peer Trusted Libraries ...................................................................................... 45  
6.4 Summary ......................................................................................................................... 46  

7 Conclusions and future work ............................................................................................. 47  
7.1 Conclusions ...................................................................................................................... 47
7.2 Future work .................................................................................. 50
Appendix A: Authentication procedures ................................................. 53
A.1 Kerberos ....................................................................................... 53
A.1.1 Description ............................................................................ 53
A.1.2 Architecture .......................................................................... 54
A.2 X.509 ............................................................................................ 56
A.2.1 Description ............................................................................ 56
A.2.2 Architecture .......................................................................... 59
Appendix B: Existing P2P systems ........................................................ 63
B.1 Napster ......................................................................................... 63
B.1.1 Architecture .......................................................................... 63
B.1.2 Security ................................................................................. 64
B.2 Gnutella ........................................................................................ 65
B.2.1 Architecture .......................................................................... 65
B.2.2 Security ................................................................................. 67
B.3 Freenet ........................................................................................ 68
B.3.1 Architecture .......................................................................... 68
B.3.2 Security ................................................................................. 70
B.4 Mangomind .................................................................................. 71
B.4.1 Architecture .......................................................................... 71
B.4.2 Security ................................................................................. 72
B.5 WebRiposte .................................................................................. 73
B.5.1 Architecture .......................................................................... 74
B.5.2 Security ................................................................................. 75
B.6 Groove ......................................................................................... 76
B.6.1 Architecture .......................................................................... 77
B.6.2 Security ................................................................................. 78
B.7 Magi Enterprise ........................................................................... 81
B.7.1 Architecture .......................................................................... 81
B.7.2 Security ................................................................................. 83
B.8 Avaki ............................................................................................ 86
B.8.1 Architecture .......................................................................... 88
B.8.2 Security ................................................................................. 89
B.8.3 Secure Grid Network Protocol .............................................. 91
B.9 Globus .......................................................................................... 94
B.9.1 Architecture .......................................................................... 94
B.9.2 Security ................................................................................. 95
Appendix C: P2P development platforms ............................................. 99
C.1 JXTA ............................................................................................ 99
C.1.1 Description .......................................................................... 100
C.1.2 Architecture .......................................................................... 101
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.1.3 Security</td>
<td>103</td>
</tr>
<tr>
<td>C.2 .NET</td>
<td>104</td>
</tr>
<tr>
<td>C.2.1 Description</td>
<td>104</td>
</tr>
<tr>
<td>C.2.2 Architecture</td>
<td>105</td>
</tr>
<tr>
<td>C.2.3 Security</td>
<td>106</td>
</tr>
<tr>
<td>C.3 Peer-to-Peer Trusted Library</td>
<td>107</td>
</tr>
<tr>
<td>C.3.1 Description</td>
<td>107</td>
</tr>
<tr>
<td>C.3.2 Architecture</td>
<td>108</td>
</tr>
<tr>
<td>C.3.3 Security</td>
<td>109</td>
</tr>
<tr>
<td>Bibliography</td>
<td>111</td>
</tr>
<tr>
<td>Literature</td>
<td>111</td>
</tr>
<tr>
<td>Links</td>
<td>111</td>
</tr>
<tr>
<td>Glossary</td>
<td>115</td>
</tr>
<tr>
<td>Index</td>
<td>119</td>
</tr>
</tbody>
</table>
Authentication in peer-to-peer systems
Chapter 1

1 Introduction
This thesis is a theory-based project that is part of the master degree education program in computer science, performed by a senior student at Linköping University in cooperation with the Swedish Defence Research Agency (FOI).

1.1 Background
Internet of today is mostly based on strict client-server communication. This has made the communication channels to, and from, some central servers to become a bottleneck in the system. The amount of on-line users today, exceeds the number of users that the Internet architecture is capable to support at once. Who has not experienced slow connections, and even worse, denial of service due to some network overload? To improve the performance of Internet communication there are at least two possibilities. One is to increase the capacity of the network and central servers. Another possibility is to develop a new architecture of the communicating network, a 3:rd generation Internet. This kind of Internet architecture will enable the possibility for direct connection between all participating nodes (peers) instead of using some centralized server to be able to communicate and distribute information between nodes. It will also reduce and spread the Internet traffic more symmetric within the network.

This report will describe the architecture of typical peer-to-peer networks and how security is provided within these environments, with focus on their authentication methods.

1.2 Motivation
If a new generation Internet will become a reality, the security and communication issues between all participating nodes must be at least as reliable and fast as the security and communication between clients and servers in the Internet architecture of today. How can this be done? The report will be able to spread some light over the security aspects within some peer-to-peer network solutions available today.

1.3 Purpose
The purpose of the report is to collect information about available solutions of how security issues are applied to peer-to-peer networks, with focus on authentication methods. The report may be used to get some
information about the alternative network technologies of today, in the aspect of building new secure network solutions.
The main purpose is to present how nodes can authenticate each other, and what kind of authentication methods that can be achieved today within existing peer-to-peer network architectures.

1.4 About this document
The report will first give an introduction to some basic areas:
Chapter 2 will describe the main definition of peer-to-peer systems.

Chapter 3 will describe security issues (e.g. authentication).

The idea with this introduction is mainly to let the reader understand what a peer-to-peer system looks like and what is supposed to be included in the aspect of security.

Chapter 4 will describe how authentication can be performed, in general, and how authentication/verification keys can be distributed in a secure manner within a network.

Chapter 5 will describe some existing peer-to-peer applications. Both their main functionality and security aspects will be discussed, with the focus on security and authentication methods.

Chapter 6 will describe development platforms (or frameworks) suited for peer-to-peer communications. The focus of the platform descriptions will also be on security and the included possibilities of authentication.

Chapter 7 will give a report summary with the author’s conclusions together with some examples of future work regarding peer-to-peer security.

An appendix will provide the reader a possibility to get more information about authentication methods (Kerberos and X.509), some existing peer-to-peer applications (e.g. Napster), and secure development platforms (JXTA, .NET, and PtPTL).

The information in this report has been obtained from specific peer-to-peer literature, articles, and technical documentation on Internet as well as in books written for peer-to-peer application developers.
The bibliography, at the end of this report, will give a complete list of all the reference literature being used in the report.
Authentication in peer-to-peer systems
Chapter 2

2 Peer-to-Peer: definitions and architecture

In a traditional client-server network, as the Internet of today, communications are based on the use of central servers. These servers offer the possibility to store information and route requests over the network. They are working as platforms on the network where clients can collect their specific results when needed. Servers are also used to provide asynchronous communication between clients, which is a typically centralized behaviour.

Figure 2.1

Servers in a client-server environment (figure 2.1) are normally unattended systems in some backroom, responding and acting on the incoming requests. Specialized personnel with good knowledge of how a network should be configured and what kind of security is needed are usually administrating the servers. These persons also use to be responsible for the security aspect of, and within, the server system.

Another solution of how to establish communication between clients in a network is the peer-to-peer model (figure 2.2), where all the participating computers are nodes (peers). This means that all the separate participating nodes will be able to interact directly with each other. The definition of nodes, in this aspect, involves both person handled machines and stand-alone machines.
Nodes are, in this model, allowed to exchange information directly with each other without accessing some central server, as in the server-client environment. It is, however, possible that P2P systems may be hybrid, which includes the use of some central server. The server will, most times, not hold any data itself. It will just hold information about where in the network data and resources may be found. Communication and exchange of information is performed directly between the participating peers. This is a typically synchronous and decentralized behaviour.

Figure 2.2

Most peer-to-peer systems also provide a possibility to add support for asynchronous file and message exchange. For example, a peer would like to send some important message to a specific peer that not is connected to the community at the moment. In this case, it will be possible to put the message on some central message relay service that will send the specific message to the receiving peer as soon as the receiving peer connects to the network environment. This service can, in fact, be applied to some participating peer in the network. It does not have to be provided by a centralized stand-alone server.

2.1 Definitions

The main idea behind P2P is that each participating peer will be able to act both as a client and as a server in the context of some applications. The peer-to-peer machines are, therefore, not rigidly defined as either clients or servers. They will have the possibility to act as normal clients in a client-server system, but they can also act and respond as a normal server.
to some other client’s requests. This is possible because of a specific middleware layer (figure 2.5) in the peer-to-peer architecture.

Peers will have the possibility to be organized in groups, without any assistance of a central authority. These self-governed communities can share, collaborate, and communicate, or participate in their own Virtual Private Network (VPN).

The P2P environment is dynamic, which means that it will grow, or shrink, when peers are connecting, or disconnecting, to the environment. It is possible in some P2P systems (for example Gnutella, described in Appendix B on page 65) that peers connected to the same network within the same community can become invisible (disconnected) from each other at any time, even though they are still on-line. This reduction of visible nodes can appear if some peer in the connecting route disconnects from the community. The community will then be divided into sub-networks. These sub-networks can, at any time, be reconnected with another sub network when some peer connects to the network community. This is possible if the connecting peer has knowledge of some peers within different sub-networks. The connecting peer will with its connections glue together the different sub-networks.

Figure 2.3
A typical client-server scenario (figure 2.3): A client (John) would like to get a file from another client (Jane):
1. John mails Jane: "I need to borrow a copy of your project file Planet.doc"
2. Jane reads the mail when she synchronizes her inbox
3. Jane replies to John: “I will place Planet.doc on the groups shared drive”
4. John reads the mail when he synchronizes his inbox
5. Jane uploads file “Planet.doc” to the shared drive
6. John downloads the file from the shared drive

It is a typically centralized, and asynchronous, scenario where the central servers are controlling all communication between the participating clients. No direct communication between the clients is performed. If, in this example, Jane does not check her mail for some time, she will make John wait for the response and delay the requested file for an unnecessarily long time. The problem may be solved with a direct connection between clients connected to the environment, as in the peer-to-peer network.

Figure 2.4

Peer-to-Peer scenario (figure 2.4): The same example as above may look like this in a P2P system:
1. Instant message, John to Jane: “I will need a copy of your file Planet.doc”
2. Instant message, Jane to John: “It’s on my shared storage now”
3. John downloads the file from Jane’s computer

The P2P solution seems to be much faster and does not demand as much traffic over the network as the pure client-server model, but it demands that the peer (Jane) is on-line if she is the only one holding the requested file. If the file also is stored on some other peer’s shared storage, the peer (John) can connect to that peer instead. Most P2P applications will support this possibility, and collects information from all shared resources within the network community when some peer searches for a specific resource. This may result in a huge list of matching resources and filenames if the name of the resource or file is common.

Most P2P file distributing applications will have a limit of matching hits to minimize the list. It can, however, be some problem to identify the correct file in the list of matching filenames. This problem has been solved in a way by some P2P file distributing applications. They organize their shared file structure in areas of subject and interest, so all (or at least most of) the files within the same subject will be located near each other. This will make a file search much more efficient. It may also give users a possibility to find related files within the same subject.

P2P networks permit peers to share more than just files and disk space within the community. The possibility to share network resources and processing power is also an important part of the P2P environment. When peers are sharing processing power with each other they can interact and solve computational tasks of great complexity and magnitude. This is called “cycle sharing”. One example of cycle sharing is the SETI@home [S1] project where over 3.5 million users (in February 2002) from over 200 countries contribute with processing time on their local computers to help in a scientific Search for Extraterrestrial Intelligence (SETI). The application is designed to run as a screensaver, which means that the participating users will only contribute with their system resources when the system is idle. Each participant will get a chunk of work that takes approximately a couple of days to process.

SETI is a scientific effort to determine whether it might be some intelligent life somewhere in the universe, by analysing data collected from billions of radio frequencies that flood the universe. The total aggregate performance within the SETI project is larger than that of the most powerful supercomputer.
The client-server model and the P2P model are not totally exclusive of each other. They can, and do, co-exist in most networks. One example is the Internet where there exist both traditional client-server systems and some peer-to-peer systems using the same basic network protocols to communicate over the network.

### 2.2 Architecture

The peer-to-peer architecture (figure 2.5) is more sophisticated than the regular client-server architecture due to the dual (client/server) function of a peer. Any peer can behave as a client in a client-server model with the opportunity to switch to become a server and respond to incoming requests from other peers at any time. This is possible because of the additional middleware layer of software in the P2P architecture.

Figure 2.5

![P2P Architecture Diagram](image)

The middleware layer interfaces to the physical layer underneath and to the applications above. It supports process and resource management, access control, and security. This creates a solid and secure platform for developers to build new secure peer-to-peer applications upon.

The P2P middleware layer contains Location Independent Services (figure 2.6) where local independency manifests itself in a couple of ways:
The services will look the same, regardless of what kind of platform it runs on, but their functionality can be limited on some platforms e.g. handheld devices with less memory.

Users will interact with objects in the same manner regardless of the objects location. The user should not be able to tell if the resources are located locally or not.

Figure 2.6

![Components of the middleware layer](image)

The capabilities needed in the P2P middleware architecture in figure 2.6, from the bottom up are [L1]:

- **Communications.** It contains common protocols like HTTP or SOAP, or a socket mechanism. It will also involve services that support communications crossing firewalls and Network Address Translations (NATs) between a local node and an Internet Protocol (IP) address.

- **Availability.** Peers are expected to exercise intermittent presences in the network, and the middleware must enable the application to deal with a number of related issues. For example, an attempt to reach a peer may fail and the application must be able to deal with that. The solution to this problem may include queuing of the message, or notification, using other peers (relay services). Another aspect is content availability. It may be expected that some content must be available and persistent over time. This will most likely
require replication of files in such a way that persistence and availability is assured.

- **Security.** Security and trust is one of the major concerns people have against P2P computing. Most P2P services are dependent of some security-related issue. The security layer will support this dependency and address a number of topics:
  - The user and the application must manage access control between all peer computers, including identity authentication, control of access rights and permissions to other parties.
  - It has to guarantee both confidentiality and privacy at levels acceptable to the users. Peers must trust the integrity and authenticity of all content they will receive over the network.

Tools like keys, digital signatures and encryption, among others, implement these security issues.

- **Identity, Presence, Community.** This layer deals with the establishment and management of identities of both individuals and groups. The identity is related to the naming scheme and to the authentication aspect of the security layer. P2P groups and communities require an identity that may have policies and common permissions associated with it. This means that membership in these groups, or communities, automatically implies acceptance of the responsibilities and privileges that are assigned to those areas.

  Another important aspect of the identity, related to security, is the communications with unknown entities. It is useful to include capabilities for attestation, and to make some statements about the user and reputation (some measure of what others think of the user). This layer also deals with the matter of presence. Presence involves the possibility for peers to indicate their network status. It will indicate if they are offline, connected but not engaged, or active in the P2P network exchange.

- **Administration and Monitoring.** This layer supports a huge decentralized organization of resources and actors. Some P2P applications seek anonymity and privacy (like in Freenet, described in Appendix B on page 68). It is, however, often a good reason to audit access and actions. Peers may want some information of how and when their resources have been used. Service providers may also require usage and identity information before releasing the services. This layer is highly application-dependent and its services
may be embedded within other services, such as resource management services.

- **Naming, Discovery, Directory.** These services are related, but well separated. The naming service deals with management of namespaces so they encompass the P2P universe. Names are also associated with groups and peer entities. Users are not the only things that need to be named in a P2P environment. All resources need to have an identity to be able to request a specific resource in the distributed P2P communities. These resources can be computers, handheld devices, printers, and so on. A naming service will provide the different resources with an identity that will be used in the authentication and authorization services. The name scheme must be given some careful thought to get a scaleable service of the millions of possible peers and resource elements.

The discovery service provides applications the means to find out capabilities and availability of resources. This is a kind of information that can be required often, and can easily be stored in a local directory for quick and easy access. A typical example of this is the indexed list of MP3 music files that Napster servers, described in Appendix B on page 63, creates for its participating users.

- **Sharable Resources.** This is fundamental within a P2P computing environment. A resource management service is most likely a collection of services for different kinds of resources, and the various aspects of managing them. It is possible to encounter a service that manages access to some shared storage. This shared storage can be an allocation given on some peer machine, or an aggregate of vast amounts, I/O services, file abstractions for a storage service, and file system semantics.

Other shareable resources e.g. a cycle sharing service can do as little as enable a guest’s executable code on a peer machine, or as much as scheduling and dispatching a computational task that spans over many computers with different capabilities and operating systems.

The middleware layer components include the standards chosen to support the structure of the P2P environment, providing common protocols and policies that need to be consistent and enforceable across the various services.
Most applications just need to use some parts of all the services provided by the middleware layer. The environment in which the application runs will also sometimes be able to determine the amount of middleware services required by the system.

It is, however, important to have good security mechanisms to be able to get a trustworthy P2P environment. Security aspects affect any communication and access performed within a P2P application. These security mechanisms must not be allowed to overwrite and compromise any of the local policies on each peer’s computer.

One solution is to separate the local peer’s private and shared areas, and let peers work and share spaces within a trusted area (or sandbox, described in the authorization section on page 21). The visiting peers and their applications will not be allowed, or able to, get hold of other resources than those specified within the trusted area.

The P2P architecture includes two methods used to establish P2P environments, the hybrid P2P method and the pure P2P method. These methods will be described further in the following sections of this chapter. No method can be told to be better than the other. It is totally depending on the circumstances around each specific P2P application. Some systems are more suited to a hybrid peer-to-peer network and others to the pure peer-to-peer network architecture. This choice is totally up to the application developers.

2.2.1 Hybrid peer-to-peer networks

Hybrid peer-to-peer networks (figure 2.7) are centralized in the sense that they depend on some central server. The server is not holding any data itself, it is mainly used to organize the network.

A hybrid P2P scenario: A user activates the P2P application and connects to the central server for authentication. The server will verify the user’s identity and collect some information about the user (the identity, the current IP address, shared resources and files). The user’s environmental status parameters will then be updated to “online”, or some similar expression of the user’s current state.
The information collected by the server is used to simplify the search procedure when a peer is looking for some resources or files. This kind of solution is used, for example, in the Napster network, described in Appendix B on page 63, where the central server holds an indexed list of all MP3 files together with information about the peers hosting these files. It is a very efficient way to search and distribute files and other resources. The server checks its indexes and responds to every request with a list of matching filenames together with peer addresses of the hosting machines. The requesting peer will then be able to select a target peer from the list and send a download request directly to the target peer.

2.2.2 Pure peer-to-peer networks

Pure peer-to-peer networks (figure 2.8) are decentralized, without any central server. This kind of system is built on participating peers only, connected to each other. No central administrator unit will be involved to distribute information within the community.

The network environment will be formed automatically when peers log into the system and establish connections to other peers. There are several solutions of how pure peer-to-peer networks may be connected. One example is the Gnutella method, described in Appendix B on page 65, which will let a connecting peer announce its presence to another client. That client redistributes this information to eight other clients that it
already has established contact with. Each of these eight clients will tell seven others, who will tell six others, and so on. This way each client will make a large number of clients become aware of its existence and what kind of resources it is sharing.

Figure 2.8

2.3 Summary

There are two main architectures of peer-to-peer systems. First, the hybrid peer-to-peer method including some central device (server) used for authentication and organization purposes of the environmental resources. Second is the pure peer-to-peer method, which consists only of participating peers and no central devices. No peer-to-peer method can, however, be told to be better than the other. It is totally depending on the circumstances around each specific application. Some systems are more suited to a hybrid peer-to-peer network, and others to the pure peer-to-peer network architecture. This choice is totally up to the application developers.
Chapter 3

3 Security: definitions

Security within network architectures is mainly a question about trust. The security solutions must be well organized so all participating entities will have faith in the security solutions. It includes solutions with cryptology to establish secure network communications and file systems. Peer-to-peer networks push this concept one step further. It is important in a P2P environment, where the separate nodes build the network structure, that the peers’ local operating system security mechanisms are secure. If not, the total network solution cannot be trusted to be secure.

The security issues are especially important in peer-to-peer systems. It is because these systems are decentralized and no central administrator is responsible for the security issues. In the P2P environment where every peer acts both as a client and a server, most of the users do not have any experience of how to support, run, or configure a server. This can open huge security holes in a system. For example, if the local administrator (peer user) by mistake opens up all local disk space for sharing instead of just one specific directory. It will mean that all other peers can get hold of sensitive information on that system and use it for malicious purposes.

Peer-to-peer networking also includes other security relevant ad-hoc moments:

- During firewall tunneling a direct connection between the requiring peer and the receiving peer may be established. The firewall software may not be able to monitor the content being transferred over the direct connection and prevent hazardous material from entering the system.
- After collecting and installing an application from some other peer, via a direct connection, it is difficult to know what kind of side-effects this application will have on the receiving peer’s local system, or what problems it may cause the local network.
- Sensitive information sent via Instant Messages (IMs) is in many applications not protected via encryption. Messages of this kind can easily be intercepted. Despite of this risk, people often use IMs to exchange sensitive information. The same security risk also exists in the regular email system. Some people will, however, believe that the IM environment really permits a secure connection between the participating peers, just because they are using direct communication with each other. This direct connection is actually
routed through a large network, like Internet, and provides possibilities for other parties to monitor the communication. Today, however, it does exist some applications providing encrypted IM communication, but they are not that common yet.

3.1 Establishment of a trusted environment
Trust in a traditional client-server environment is related to the central servers in the system. All clients have to trust the servers to be able to rely on the information they may get from the servers.

Building a trusting environment within a peer-to-peer system includes a great deal of planning. P2P security includes a lot of security issues to be taken into consideration and it should at least include the following parts:

- Provision of a possibility to integrate with other local security measures and adapt to their local policies without compromising any local settings.
- Provision of a framework for access control to enforce authorization policies.
- No exposition ever of information and data of others to any unauthorized users. Delegation of tasks and authority has to be adapted to the rules set by the owners of the identities involved. Secure data has to be protected during the transfer, if it is going to be sent to some other location.

In a P2P environment all nodes must be able to trust each other. This is mostly performed via some trusted certificate authority or some other trusted central authorities. Most applications in this report may use trusted central authorities to create a secure and trusted environment. To be able to get these environments more scalable the authorities must be organized in some kind of hierarchy with a top-level authority, which most likely is a primary initiated authority in the system.

There are, however, other approaches towards the goal of trust in a system. Some applications are restricted in the way they may operate and their main purpose may not include any strict security. In most systems is security a main issue. This includes the matter of trust between all participating objects. Trust may be ordered in levels where very trusted objects may be objects certified by some central trusted authority. Non-certified objects that are assumed to be trusted belongs to a lower level of trust, which may include objects without centralized authorities.
The trust is, in this case, distributed in some way by other known (and trusted) objects.

There also exist some ideas of creating rumour-measurements in some systems that will give participating objects an indication if other objects in the environment may be trusted or not. It is based on human-like behaviour where it is easier to have faith in some object (or person) that many other objects (or persons) trusts than some unknown object (or person). This is a very low level of trust that may be difficult to use in a real system.

### 3.2 Authentication

Authentication is the process of verifying the identity of an entity. It is also in a peer-to-peer system a verification that the node some peer is communicating with really is who it claims to be.

Authentication is more difficult to achieve in a pure P2P network architecture than in a centralized environment. This is because no central server is used to verify a peer’s identity. Authentication must be performed within the system between each pair or group of peers. It is, however, possible for pure P2P systems (as well as hybrid ones) to use some third party (a central authority) in the authentication process. The central authority may be used to generate unique initial global identities for peers and resources, or to generate and distribute certificates or tickets, described in chapter 4, to assure secure methods of authentication.

An authentication process is divided into two main parts, identification and verification [L1]:

- Identification is the procedure where an identifier is presented by the entity to the security system. The identifiers shape and form depends totally on the security mechanism used within each system.
- The verification procedure follows directly after the identity has been issued. It is where the entity presents some evidence, which proves that the identity is correct. The included evidence must be validated before the authentication sequence can be completed.

After a completed authentication, the identities presented are used as a basis for other services where security is important.

There are different levels of authentication depending on what kind of environment it is supposed to be applied to. Of course, the more demanding the verification process is the more confidence it provides.
stricter authentication comes with a cost of higher overhead and added complexity. This is why the authentication methods should match the value of the data being accessed. It should also correspond to the type and level of access the user is providing. For example, within a trusted environment e.g. a company’s local intranet there is no need for a highly advanced authentication method. This will prevent any unnecessary overhead within the community.

If, on the other hand, someone outside this trusted environment, for example from Internet, would like to access some resource within the trusted network, there is a need for some more trusted and stricter, authentication methods. This is always necessary when entities from not trusted environments would like to have access to resources within a trusted environment.

The most common way of categorizing authentication methods for users is by the concepts of “something you know”, “something you have”, and “something you are”:

- **Something you know.** It is, typically, a username and a password. Many client-server systems use this method for authentication on Internet. It is a one-element method and is considered to be quite weak. It is not a good choice for a peer-to-peer application because the user must either use a different password for every peer he, or she, wants to access, or they must give some peers a password they can use to impersonate them elsewhere.

- **Something you have.** In the next level the user will add “something you have”. This can be a smartcard accompanied by a Personal Identification Number (PIN), which is part of “something you know”. In a networked environment the nodes may have a digital certificate and know a key, and the object they have in their possession can also have a limited Time To Live (TTL). This is called two-part authentication and it is the most commonly used method in peer-to-peer security systems.

- **Something you are.** This is not only knowledge of information, but also knowledge of biometrics. The “something you are” is something unique for every individual in the world. It can be a voice, a handprint, an iris scan, a fingerprint, and so on. This kind of verification will, however, require more sophisticated equipment and a greater amount of processing.
Authentication opens up the possibility to let just authorized entities to have access to data content and applications in a system. It will also include the option to check who did what, and when, if needed.

Lack of standards in P2P communications (so far) is one of the main reasons for the existence of several “local” communication and authentication solutions. Most applications have their own solution of what the communication and authentication method will look like to suit their typical application environment.

All local authentication solutions are mostly small modifications of general and well-trusted authentication methods used in regular client-server environments.

One solution of how authentication can be applied in P2P environments includes the use of private/public keys, described in chapter 4 on page 28. The main problem with public keys is to initiate new peers and distribute public keys within the community in a secure and trustworthy manner.

Another solution to implement trust into a community is to add some kind of reputation method where a measuring mechanism can indicate if the peer, in question, seems to be reliable or not. All network communication includes the question of trust and if someone has a reputation of being trustworthy some other peer also may be able to trust this entity.

### 3.3 Authorization and access control

Authorization\(^1\) is the process of giving someone permission to do, or to have access to something. This “someone” can in the P2P environment even be an object. It controls when and how entities will be able to access any local resources.

The authorization process itself is not quite as tricky as the authentication process. It does not necessarily involve any communications over the network. Generally, the authorization process uses an Access Control List (ACL) that contains the permissions granted to different entities in the system. In these situations the authorization process is to check the current request against the ACL. If the request falls within the parameters of permissions granted the request will be granted. If not, the request will be denied.

Another possibility to track permissions is by placing them in the resource objects. This is a quite natural approach when all resources are treated as

---

\(^1\) In this report will the definition of authorization be the answer, yes or no, to specific requests. Access control will, on the other hand, be the more resource-specific decision of who will be able to require specific resources and with what privileges. These concepts can, however, be defined to have the opposite meanings in some literature.
objects. The object will then contain information about users, other objects, groups, and whether or not they may request the resource. This kind of authorization will be a part of the call to every object-resource.

The procedure of managing access control is more difficult, and demand more consideration and effort. It is a major procedure to choose what kind of rights each resource should have in different situations and at different times.

Authorization and access control procedures are closely linked together, but they have quite separate functions [L1]:

- **Authorization**, in the context of this discussion, is a specific operation. It is the act of granting or denying access in response to specific requests.
- **Access control** is the mechanisms and policies that restrict access to computer resources, in a process.

Every participating node in a P2P environment will have to take care of the access control issues in their own local system. To be able to provide a reliable access control mechanism each local system should at least answer to:

- Who should be able to have access to the system?
- Which of the local resources will be available, and to whom?
- How can times, when others are allowed in the system, be restricted?
- How can visitors, in the local system, be disconnected, or denied access to some local resources, at any time?

The authentication and access control methods must work within a very dynamic environment to be able to support P2P networks. Users, the permissions granted to them, the resources, and what every single user allows in their local system, will constantly be changing. Something or someone must manage these environmental changes in the system. The P2P infrastructure, or application, must take care of the access issues within the system. It must decide who will belong to the community and who can make requests of the community resources. The application must also define all shared files and what kind of privilege the users will have on these files. This will work fine in a trusting environment, as a company’s local intranet, where authentication and the access mechanisms assure good
behaviour for its users. It is, however, a bit more difficult in a global network, as Internet, where the environment cannot be trusted.

To be able to let less trusted nodes to enter and access resources and files in some peer’s shared space there should be some possibility to securely separate the shared and private spaces. One solution of the separation process is to use sandboxes. The sandbox is a secure area where entities can work with shared resources, while the rest of the system is off limits. The main items monitored are:

- File access, including network-connected drives. The sandbox software will verify permissions to access files and the operations allowed there (read, write, copy, delete).
- Operating system calls. The application, in general, is prevented from modifying any settings.

There are some different methods available to implement a secure sandbox concept. The main methods are [L1]:

- Sandbox boundaries, which can be defined within the context of the application and the behaviour of the application is constrained, or encapsulated in some way to prevent access to non-shared resources.
- The Virtual Machine (VM) technique will provide a general-purpose sandbox. It is a complete set of software components that make up an Operating System (OS) and its main utilities. The sandbox, in this case, is a complete operating system sitting on top of another. This method provides a secure separation of the participants in the P2P network and the private desktop environment. The P2P software can occupy one OS and the private desktop the other OS, which will separate the private and shared spaces in a secure manner. P2P applications will be able to make system calls and modify OS settings because the OS is located within the sandbox.
- The third type of sandboxing is similar to the capability provided for Java programs via the Java Virtual Machine (JVM). JVM is a layer that interprets Java bytecode for each specific computer platform. The sandbox in Java is the program area and set of rules that programmers must use developing new Java applications. JVM mediates access to system resources and ensures a restricted sandbox. Recent versions of the Java Development Kit (JDK) are also designed to allow several levels of trust for programs operating...
Authentication in peer-to-peer systems

in the sandbox. The more trust bestowed on the code itself the more capability it has to play outside the original sandbox.

Sandbox implementations already exist in some peer-to-peer environments. For example, Groove’s shared space, described in Appendix B on page 76, confines its peers in a way that isolates them from resources that are not shared. Its sandbox functionality is closely related to the one the JVM sandbox provides.

3.4 Data integrity

Data integrity means that no data has been changed, destroyed, or lost in an unauthorized or accidental manner. It also includes that no information has been modified or destroyed in an unauthorized manner. It should not be possible for some not authorized individual to tamper with the data, neither in a shared space nor during transmissions over the network. Data integrity deals only with the constancy of data values and not with the quality of the information that the values represent.

Digital signatures provide a way to detect any data tampering. If a message is signed with a private key the recipient will know if the body of the message has been modified during transmission. A message can, naturally, be encrypted and not digitally signed.

If a high data transfer rate is important it may not be a good idea to use digital signatures because public key encryption is very slow. A symmetric encryption solution may be better in this case.

If a message is strongly encrypted, e.g. with DES, described in [L3], it is highly unlikely that the content of that message can be read by anyone, if intercepted. If someone, on the other hand, tries to tamper with a strongly encrypted message, it is unlikely that the decryption will produce anything but a meaningless stream of bits and characters. There exists, however, a way to notice if a data packet has been tampered with during transmission over some network. It is by using an Internet Protocol Security (IPSec) protocol that includes the authentication of packets by the sender to ensure that the data has not been altered during transmission.

Recovery of altered messages is not always an easy task even if it is possible to detect altered packets. It is very unlikely that anyone can find out the real content of a modified encrypted message. In most cases the receiver must alert the sender and request a retransmission of the modified packet.
Another potential risk of receiving incorrect results involves the distributed computational quality. This problem arises from the unknown hardware on which the computations are running. Different hardware architectures and system structures can give different computation results. Some old libraries and floating point rounding may affect computation results in various ways. Such variations can be crucial to the final outcome of the computation.

There are two main methods to deal with the risk of incorrect results:

- First, duplicating the computations, if possible on machines with different system structures. This means that the task has to be performed at least twice to be able to detect if the results are sensitive to hardware and software settings, or not.
- Another approach is to attach some tests to the tasks being sent over to the receiving, and computing, peer machine. If these included tests do not produce the expected results the machine in question will not be used in the future for that kind of computations.
Chapter 4

4 Network authentication methods

There are mainly two approaches used for authentication and encryption within a network. One approach uses symmetric algorithms with shared private keys and the other approach uses asymmetric algorithms with public keys. These algorithms are used both in regular client-server environments as well as in peer-to-peer networks. This chapter will give a description of how these authentication methods are defined, but first a description of how keys can be distributed securely between entities over some network.

4.1 Key distribution

Key distribution makes it possible for participant parties to exchange encryption and decryption keys. This will make it possible for participants to establish a secure connection between each other. The technique, in which keys are distributed, must be secure and well trusted by all parties. There are, however, some differences in the distributing procedures depending on if it is a shared private key or a public key.

4.1.1 Private keys

Two parties A and B can achieve shared private key distribution in a number of ways:

1. The key can be selected by A and physically delivered to B.
2. A third party can select the key and physically deliver it to both A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other encrypted by the old key.
4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to both A and B.
5. They can create a shared private key together (e.g. Diffie-Hellman).

Option 1 and 2 call for manual delivery of the shared key and is not suited in most large network solutions. Option 3 provides a possibility to send new keys over an encrypted connection between the parties. If, however, an attacker succeeds to get hold of one key, then all subsequent keys will also be revealed. Option 4, on the other hand, provides a solution including a third trusted party. This trusted Key Distribution Center (KDC) generates keys and distributes them to pairs of users over a secure connection. Each user must
share a unique master key with the KDC to be able to get a secure connection. The master key will also give the user a possibility to verify that the new session key really originates from the KDC. **Option 5** provides the possibility to create a shared private key without involving a third party. In the Diffie-Hellman algorithm, the two participants (A and B) will agree on a large prime \( n \), and \( g \) such that \( g \) is primitive mod \( n \). The procedure to create the shared key is [L3]:

- A chooses a random large integer \( x \) and sends: \( X = g^x \mod n \) to B.
- B also chooses a random large integer \( y \) and sends: \( Y = g^y \mod n \) to A.
- A computes: \( k = Y^x \mod n \)
- B computes: \( k' = X^y \mod n \)

Both \( k \) and \( k' \) are equal to \( g^{xy} \mod n \), which is the shared private key that both A and B have computed independently.

There exist an important issue about who really is the originator of the parameters in use when no courier is involved, or no established channel exist between the nodes, or without the use of a trusted third party.

The use of a KDC imposes the requirement that it can be trusted and protected from subversion. This can be avoided if the key distribution is fully decentralized. Full decentralization is not practical for larger networks only using conventional encryption. A decentralized approach requires that each end system is able to communicate securely with all potential partner end systems to be able to provide a secure session key distribution.

### 4.1.2 Public keys

Several techniques have been proposed during the years to distribute public-keys. These proposals can be grouped into the following general schemes:

1. Public announcement.
2. Publicly available directory.
3. Public-key authority.
4. Public-key certificates.

**Option 1.** Public announcement provides any participant to send its public key to any other participant or broadcast the key to the whole community. This approach is convenient, but it has a major weakness – anyone can forge such a public announcement.

**Option 2.** Publicly available directory provides a greater degree of security by maintaining a dynamic directory of public keys. Some trusted
entity or organization must be responsible for maintenance and
distribution of the public directory.
This scheme is more secure than individual public announcements but it
has still some vulnerabilities. For example, if an attacker succeeds to
access the authority’s distribution directory, the attacker will be able to
pass out counterfeit public keys. It is also vulnerable to “man-in-the-
middle” attacks where keys can be altered during their network transport.

**Option 3.** The use of a public-key authority will include stronger security
by providing tighter control over the distribution of public keys from the
directory. A central authority must maintain a dynamic directory of public
keys of all the participants. To improve security, each participant reliably
knows a public key for the authority and only the authority knows the
corresponding private key. This provides the possibility to create
authenticated communications between users and the central authority.
The central authority can become somewhat of a bottleneck in the system.
Every user must apply to the authority for a public key to every other user
it wishes to contact. This can create a network overload in large networks,
with lots of users.

**Option 4.** Public-key certificates are an alternative approach where
certificates can be used to exchange keys between participants without
contacting a public-key authority. It is as reliable as if the keys were
obtained directly from a public-key authority. Each certificate created by
the Certificate Authority (CA) contains the public key and id given to the
participant with the matching private key and is signed by the CA’s
private key.
A participant distributes its public-key information to other users by
transmitting its certificate. Other participants can verify that the certificate
really originates from the CA and trust that the included key belongs to the
identity included in the certificate.

Once public keys have been distributed or become accessible, secure
communication that prevents eavesdropping, tampering, or both is
possible. However, it is not such a good solution to exclusively use public-
key encryption for communication because of the relatively slow data
rates that can be achieved. Public-key encryption is more reasonably
viewed as a secure method for distribution of symmetric keys instead of
being used for conventional encryption.

**4.2 Symmetric algorithms**
The same key is used for both encryption and decryption in a symmetric
algorithm. The key is shared by the users (A and B), which would like to
authenticate and communicate with each other securely. Symmetric keys are usually generated and distributed by some KDC and must be kept secret. All parties sharing the same key are able to read each other’s encrypted data. Private connections to different parties will need a new key for each connection (session keys).

The KDC shares a master key with each participant. This will provide a secure distribution of the temporary session keys. A nonce is included in the message to verify that it is not a replay message and to authenticate the other party of the communication link.

Figure 4.1

Description of an authentication session (figure 4.1) [L3]:

**Step 1.** The authentication exchange is initiated by A, who generates a nonce ($N_a$) and sends it plus A’s identifier ($ID_A$) to B. The nonce will, in step 3, be returned to A in an encrypted message that includes the session key, assuring A of its timeliness.

**Step 2.** B will alert the KDC that a session key is needed. The message to the KDC includes B’s identifier ($ID_B$) and a nonce ($N_b$) created by B. The nonce will, in step 4, be returned to B in an encrypted message that includes the session key, assuring B of its timeliness. B’s message to the KDC also includes a block encrypted with the secret key shared by B and the KDC. This block is used to instruct the KDC to issue credentials to A. The block specifies the intended recipient of the credentials, a suggested expiration time for the credentials, and the nonce received from A. In this stage B is authenticated by the KDC and the session key can be distributed to A.
**Step 3.** A receives a message from the KDC, including B’s nonce and a block encrypted with the secret key that B shares with the KDC. The block serves as a “ticket” that can be used by A for subsequent authentications. The KDC also sends a block to A, encrypted with the secret key shared by A and the KDC. This block verifies that B has received A’s initial message ($ID_B$) and that this is a timely message and not a replay ($N_o$), and it provides A with a session key ($K_s$) and the time limit on its use ($T_b$).

**Step 4.** A will transmit the ticket to B together with B’s nonce, the latter encrypted with the new session key. The ticket provides B with the secret key that is used to decrypt $E_{K_d}[N_b]$ to recover the nonce. The fact that B’s nonce is encrypted with the session key authenticates that the message came from A and is not a replay.

Another similar solution to this problem is the Kerberos ticket system, described in Appendix A on page 53.

### 4.3 Asymmetric algorithms

In asymmetric algorithms, also called public key algorithms, different keys (public/private keys) are used for encryption and decryption. One key can be made public, but the other key has to be kept private. The two keys are, naturally, algorithmically related, but it should not be feasible to derive the private key from its public counterpart. Asymmetric algorithms will use each participant user’s public/private key for authentication, data decryption/encryption or digital signatures.

Public key encrypted messages are only readable by the user carrying the corresponding private key.

Private key signature in cooperation with the user’s identity is used to identify and authenticate a user. The user in possession of the private key is the only one that can make a specific signature. Everyone with access to the corresponding public key can verify the signature (and its originator). In this aspect are the keys also known as signature-, and verification-keys.

Public key cryptography makes key management and distribution easier than the management and distribution of symmetric keys. After all, public keys are supposed to be public and need no specific protection. There must to be some reliable source linking user identities to their cryptographic keys in order to make other users to have trust in the distributed public keys.
Public-key ciphers and signature schemes often rely on certificates to generate a secure connection. A CA will guarantee the link between a user’s identity and cryptographic key by signing a certificate that contains a participant’s username, public key, name of the CA, expiry date, etc. There are several proposals specifying the precise format of a certificate. These proposals differ to some extent in the fields they include and the length of individual fields. One frequently used certificate standard is X.509, described in Appendix A on page 53.

To let participating nodes verify a certificate provided by a CA, a matching (public) verification key is needed by all nodes. This key is needed by all nodes to be able to verify the CA’s signature. Certificates must not just work in a plain environment. They can also be configured to work in a hierarchical environment where one CA’s verification key may depend on another CA, and so on.

There must also exist some possibilities to revoke a valid certificate when needed. This is mostly performed by the use of special Certification Revocation Lists (CRLs) generated by the CA. The CRLs are located together with the certificates and should be checked every time some peer will get a communication request from another peer. It is, however, not done all the time. Sometimes CRLs are cached on some local machine and are checked there and synchronization with the CA’s CRLs is performed periodically. If the peer’s certificate is not included in the CRL it is most likely a valid certificate. The peer can then verify if the certificate really originates from the trusted CA, by using the CA’s public key, and also check that the certificate’s Time To Live has not expired.
Chapter 5

5 Peer-to-Peer authentication

This chapter will describe technologies and procedures of how authentication methods are applied in some peer-to-peer systems of today. The following sections will give a description of some existing P2P applications and their authentication methods.

5.1 Napster

Napster was the application that made most people aware of the possibilities with a modern P2P architecture. It has been designed to let people exchange music, MP3-files, with each other over some network e.g. Internet in a cheap and efficient way. The Napster application provides a service that does not require any encryption or strict security (so far). This is partly due to the absence of viruses in the MP3-format. Another reason is the way the application will be used. The application is used to distribute and share music files with other Napster clients and this is a service that no one seems to be willing to tamper with (so far).

Napster is a hybrid peer-to-peer system with a central server. The main purpose of this central server is to keep an index of all music files and the participating users currently online. The server does not store any music-files, it is only used to simplify the search for specific music files. It will distribute a list of locations where some requested music file might be found. This provided information lets users establish a direct connection to each other. All music-files are located on each participant’s local machine in a special directory shared within the Napster community.

The authentication mechanism within Napster is basic. Users are identified with a nickname and authenticated by a personal password. These identities are also used to organize the central index list on the server. For further authentication MD5, a one-way hash function, will be used to send digital signatures included in protocol messages. No encryption mechanism is involved in the connection between clients and servers (so far).

See Appendix B on page 63, for more information about Napster.
5.2 Mangomind

Mangomind is a Microsoft Windows file sharing system. It allows multiple users to simultaneously access, share, and store files from all over the world in the exact same manner as if the files were on their local PC.

It is a hybrid peer-to-peer environment where the central part is the Mangomind server. The central server provides storage for shared Mangomind drives, which are only accessible to invited clients authenticated by the Mangomind application.

Every invitation includes a specific password, which must be used to be able to access the shared drive. A user is invited to a drive, and verified (by the server) to have a valid invitation-password. The client’s application generates a private/public key-pair and sends the public key to the server (using HTTPS) for further authentication.

The application’s authentication process is using mutual authentication where both the server and the client authenticate each other. The host server uniquely identifies clients by their Mangomind username and the specific identifying Mangomind information stored on their local machines. This is performed automatically each time a user connects to a Mangomind drive.

See Appendix B on page 71, for more information about Mangomind.

5.3 WebRiposte

WebRiposte is a web-enabled peer-to-peer messaging and data management solution. It is mainly developed to provide a solid base for e-business applications, but it can also be used elsewhere.

The system provides a secure foundation, which can be used to build and develop new distributed applications upon. This means that developers only need to focus on the application layer to produce new reliable and scalable applications.

It is a hybrid peer-to-peer application that uses central servers to share information and to locate information requested by some participating user.

Authentication methods and management of local security policies within WebRiposte are based on close integration with the Windows NT/2000 security model. The provided data encryption mechanisms are supported via close integration with the Windows CryptoAPI. This provides a
possibility for all participating servers and clients to authenticate all of
their neighbours.
Authentication and data encryption is used to provide secure transmission of
data throughout the system. WebRiposte also uses digital signatures to prevent any data tampering. These digital signatures will automatically be verified by the application every time data content is retrieved from a server.

See Appendix B on page 73, for more information about WebRiposte.

5.4 Groove
Groove is a Windows-based Internet software solution designed to establish direct connections between participating peers. It provides the possibilities to talk, chat, send instant messages, draw pictures, exchange photos and files, play games and browse the Web together with other peers.

It is a hybrid peer-to-peer platform where all participating peers are communicating directly with each other. It uses some centralized units, called server-bots, to reduce the administrative overhead and to provide support for storage of customer history, transaction processing, and records/document management.

Each user in a shared space is allowed to collaborate with all other peers within the shared space by using signed communication.

The authentication methods included in Groove are based on public keys and/or biometrics.

Users will establish initial trusted relationships via the exchange of invitation cards, in the form of “vCards”. The vCard includes information about the peer and its public key. When a user accepts an invitation, often sent via email, he or she automatically also has accepted the sender’s vCard and public key. The receiving peer must reply by sending its vCard in return, which is done automatically within Groove. The receiver must also be able to verify the authenticity of the invitation. This can be done via digital fingerprints or voice messages.

The digital fingerprint is a hash of the sender’s public key and will be computed by the application on the recipient’s local machine the first time it receives an invitation from a sender. The user may verify the sender’s fingerprint in one of the following ways: In person or over the phone with the sender, or via some other trusted media or user (already in possession of the sender’s fingerprint). If these digital fingerprints match the receiver
knows that the invitation is authentic and may store the fingerprint as part of the sender’s vCard in the local account. The fingerprint may also be used to verify subsequent invitations from the sender. This kind of distribution of public keys ensures security without requiring any centralized certificate and key management servers.

Another solution provides a biometric authentication via voice annotation, where Groove makes it possible to emulate the way people authenticate each other in real life. For example, when a user receives an invitation from some other user, he or she, authenticates the user via a recorded voice message.

To make this authentication method more secure it is also possible to add some personal shared secret. This approach uses all human forms of recognition, as opposed to the typical methods used by public key infrastructure-based systems in which a peer always trusts that someone notarises that every peer is who it claims to be.

See Appendix B on page 76, for more information about Groove.

5.5 Magi Enterprise

Magi Enterprise is developed to give all participating peers the possibility to communicate and interact with each other in a secure manner. It provides chat and messaging capabilities on any Internet-enabled device, and support for multi-way communications between servers, PCs, PDAs, Internet-ready phones, and embedded chips.

It is a pure peer-to-peer application using open standard protocols to create a secure cross-platform environment for collaborative-intensive applications. The security provided by the application links teams together to permit file sharing, file searching, instant messaging and chat. Magi can also be used to establish VPNs between participating groups of peers.

Each Magi peer generates its own RSA key pair during the peer’s initiation and sends the public key along with its username and password to the central Magi CA. A password is used as verification for who may apply, as well as subsequent authentication in case of a compromised key. If the password is accepted, the CA will bind the username to the public key with a digital certificate.

This method provides the definition of a unique name space that can not only use standard human readable names, but also correspond to an existing name space.
One of the most frequently used authentication methods in the communication between Magi peers includes the use of X.509 Public Key Infrastructure (PKI), described on page 56, governed by a Magi CA in combination with HTTP content encryption (SSL). The access control and authorization procedures include the use of ACLs. This combination allows security certificates from recognized X.509 CAs to establish the true identity of any Magi-enabled peer device, which ensures that peers only will communicate with known authorized peers.

Magi does also provide support for some other alternative authentication methods [L1]:

1. **Basic authentication.** Basic authentication demands that every request is accompanied by a username and password. One advantage with this method is that all web browsers support it.
2. **MD5 authentication.** The MD5 authentication provides a more secure challenge/response mechanism than the Basic method.
3. **Tokens.** This is, together with SSL certificates, the most commonly used authentication mechanism in the Magi environment. It is a Kerberos derived security system called “Token Authentication”. Like in Kerberos, on page 53, Token Authentication is a centrally managed authentication mechanism that authenticates peers using Magi’s Dynamic Name Service (MDNS). After authentication with the MDNS direct communication between two Magi peers is possible, but they must first obtain a session key from the MDNS. The session key is shared between the participating Magi peers and the MDNS during the whole active session.
4. **SSL Certificates.** SSL certificates are defined as a part of the HTTPS standard. With this policy the MDNS acts as a trusted third-party and a CA issuing a unique, tamperproof certificate to each Magi peer. To get good efficiency Magi peers cache certificates in a private key ring to avoid making repeated requests to the MDNS service. This kind of key ring also has the advantage that it can be consulted when the MDNS is unreachable, such as when two peers are using a private network without access to the Internet.

See Appendix B on page 81, for more information about Magi.

### 5.6 Avaki

Avaki has been developed to provide all forms of P2P computing. It is an integrated platform that runs on all major operating systems. The vision
for the product is to create an illusion and the benefit of a single virtual machine that encompasses devices from supercomputers to handheld devices and incorporates diverse architectures and operating systems.

It is a pure peer-to-peer application based on comprehensive grid software that enables efficient and secure sharing of data, applications, and computing power. The application also enables the possibility for the collection of resources to be used as a single virtual operating environment.

The main goal of Avaki’s security approach is to eliminate the need for other software-based security controls, substantially reducing the overhead of sharing resources. The authentication procedure within Avaki is decentralized and it does not require any use of a trusted central authority, even though such an authority may be useful in the process to establish initial peer and resource identities. Avaki has, however, recently (in February 2002) presented a draft for a new security protocol (SGNP) where the identities not necessarily must be based on central authorities, they may also be generated locally. The SGNP is using a Grid Naming Service (GNS) and a Resolver hierarchy where all clients only know about one, or more, GNS but has no knowledge about the Resolver hierarchy. All client-requests will go to a GNS, which will use the Resolver hierarchy to find the correct binding to the requested resource and return this binding to the client. There seems, however, not to be any direct connection between different GNSs. This will make the knowledgebase of every GNS very large to be able to support bindings between resources in many different domains.

The authentication method in Avaki is based on all object’s unique identity by using a PKI-based public key identifier through which each resource will be authenticated automatically. The public key is included as a part of each object’s name. This way objects are able to use the public key of another object to encrypt their direct communications. Likewise, an object’s private key can be used to sign messages, providing authentication and non-repudiation. The integration of public keys in every object namespace makes Avaki avoid the need for certificates to establish secure direct communication between objects.
If an intruder tries to tamper with the public key of a known object it will only result in the creation of a new object name, unknown to the community.
Avaki also includes support for authentication via Kerberos tickets, described in Appendix A on page 53, and also X.509 certificates, described in Appendix A on page 56.
It also supports secure communications using SSL encrypted connections.

See Appendix B on page 86, for more information about Avaki.

5.7 Globus
Virtual organizations tend to be fluid, which means that the authentication mechanisms must be very flexible and lightweight to allow administrators to quickly establish and change resource-sharing arrangements. Globus uses Grid Security Infrastructure (GSI) to enable secure authentication and communication to meet these requirements.

A central concept in the GSI authentication process is the use of certificates. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating peers or services within the environment.
Authentication is a central part of GSI, together with message protection. It defines single sign-on algorithms, protocols, and cross-domain authentication protocols. The methods are based on a PKI and uses authentication credentials composed of X.509 certificates, described in Appendix A on page 56, and public keys, described in chapter 4 on page 28.
GSI builds on and extends the TLS protocols to address most of the issues listed above. The main authentication issues include, besides single sign-on delegation, also integration with various local security solutions. These solutions include the use of Kerberos, described in Appendix A on page 53, and user-based trust relationships.

In brief, a Globus peer generates a public/private key pair and obtains an X.509 certificate from a trusted CA. All peers can use their X.509 credentials to generate temporary credentials, which can be used to access resources at multiple sites without repeated authentication.

See Appendix B on page 94, for more information about Globus.
5.8 Summary

There are many P2P applications available today and some more are under construction. Most of these applications do support security and the possibility to verify other peers’ identity (authentication) within the same community. The applications also support authorization and access control, which means that they are able to control what kind of resources and privileges a visiting peer will be given.

Authentication in peer-to-peer applications like those described in this chapter build on both pure and hybrid peer-to-peer architectures. Hybrid peer-to-peer applications with some central device (server) will, sometimes, use the server to authenticate all the connecting peers. All peers must, however, also authenticate each other to be able to establish trusted direct communications between peers. Most of these peer-to-peer applications will use some kind of PKI (including certificates) in their effort to authenticate each other. Another common solution is the use of tickets (like Kerberos) to provide authentication between all participating peers.

Groove does, however, have their own solution of how to distribute information about the client (including public keys) within the community. They will let each peer client distribute client information and public keys included in vCard invitations, which they can send to any other peer in the community. This solution will give participating peers a good possibility to exchange public keys within smaller groups without using some central CA.

There are also some differences in the methods pure P2P applications will use to provide authentication.

The Magi system uses a central Magi CA (MDNS), which supports the environment with tokens (like in Kerberos) as well as certificates. This solution will provide a secure and trustworthy procedure that will allow peers to authenticate each other. It naturally also requires that all participating peers must have trust in the CA providing these services. Globus uses a similar solution as Magi and lets a trusted central CA provide security for the distributed certificates, which includes each peer’s identity and public key. The architecture and system solutions of Globus and Magi are, however, not at all the same.

Avaki uses another procedure to let peers authenticate each other. Each object’s public key is included as a part of its unique name. It makes all peers aware of each other’s public key, which provides the possibility to establish a secure connection. It will also provide a method to authenticate
messages from other peers without using a central authority. This is done by the use of public and private keys. A message will be signed by the sender’s private key and encoded by the recipient’s public key. This way, only the correct recipient may read the message (authenticity of the receiver) and the signature will authenticate the sender. A central authority may, however, be necessary in the process of generating each object’s initial, and unique name. One interesting aspect of the use of central authorities involves the new protocol (SGNP) that will, according to Avaki, let the system generate unique global identities locally and provide a very scalable environment.

If anyone tries to tamper with some Avaki object’s public key (or name) it will only result in a new object identity, unknown to the community.

Groove and Avaki provide authentication solutions that do not need any central authorities. They do, however, have some potential problems in their effort to eliminate the central authority.

The procedure to get hold of an authentic reference fingerprint in Groove (to verify the digital fingerprint of an invitation) is a problem in absence of voice messages and personal delivery.

Avaki’s new SGNP may also have some potential problem in the way they let different Grid Naming Services have knowledge of the same Resolver Domains.

Another question that still remains unanswered is what kind of method Avaki uses to distribute knowledge of available resource identities within the environment? To create a secure authentication solution the distribution of identities must be trusted by all parties.
Authentication in peer-to-peer systems
Chapter 6

6 Secure P2P development platforms

This chapter will give a description of some of the more popular peer-to-peer development platforms to date. The description will give a brief overview of the different platforms, their functionality and architecture. The main issue will be to describe what kind of security they permit, and in particular what authentication methods they provide.

6.1 JXTA

The JXTA project started in the summer of 2000 by Sun Microsystems as a P2P research project. Even though Sun was behind the initial project, JXTA is now an open-source initiative and not a proprietary effort by a single vendor.

Project JXTA’s main idea has been to develop a secure peer-to-peer platform only using open standards and protocols such as Java and XML. This technology allows communication between a community of peers, or peer groups, where peers can establish secure direct communication with each other.

The peer discovery mechanism can be completely decentralized, completely centralized, or a hybrid of the two, which is important to be able to support all kind of P2P networks.

All on-line users in a JXTA environment will automatically be part of the global group “NetPeerGroup”. Communication and exchange of information between peers may be performed in this global group or in other user-defined groups. The security toolkit includes TLS at group level to provide secure communications from end-to-end and to be able to ensure secure communication within groups. All group-owners will have the possibility to select who will have access to their specific group. This provides the group-owner to allow access for any user, or restrict group membership to only those peers that are authenticated.

The authentication methods in the JXTA framework are modelled after the Pluggable Authentication Module (PAM). This structure provide methods that can be used to integrate login services with several different authentication technologies:

• RSA
Authentication in peer-to-peer systems

- DCE
- Kerberos
- S/Key
- Smart card based authentication systems.

No authentication method is more likely to be used than any other. The choice is totally up to each application developer. JXTA also provides support for a basic, password-based, login scheme that also can be plugged into the PAM framework.

Other security issues provided by the security toolkit include a library supporting cryptographic mechanisms:
- Hashing functions (e.g. MD5)
- Symmetric encryption algorithms (e.g. RC4)
- Asymmetric crypto algorithms (e.g. RSA).

Another cryptographic mechanism is the transport security mechanism that is modelled after SSL/TLS, but without the possibility to perform a handshake, or a two-way authentication on a single pipe, as a result of the use of unidirectional pipes in JXTA.

See Appendix C on page 99, for more information about JXTA.

6.2 .NET

The .NET Framework, developed by the Microsoft Corporation is a software development framework for web applications. The framework is mainly focused on the traditional client-server model, but it also provides services that can build a platform for secure peer-to-peer applications.

The framework gives developers and administrators control over their applications and resources security at a high level. It also provides an easy-to-use toolkit to implement powerful authentication, authorization, and cryptographic routines.

.NET provides support for several authentication mechanisms, which can be used both in P2P applications as well as in pure client-server applications. The framework includes the following main authentication methods [L5]:
- **Basic Authentication.** It sends the username and password to a web server in plain text. Microsoft's Internet Information Server (IIS) will authenticate the login against the database of users for the specific domain.
• **Basic over SSL Authentication.** It is similar to the Basic Authentication, except that the username and password are transmitted over a secure connection, using SSL encryption.

• **Digest Authentication.** It is a hashing technique that sends client credentials securely to the server.

• **Integrated Windows Authentication.** It is used for intranet scenarios only. It uses the client’s Windows login information for identity verification.

• **Client Certificates Authentication.** It requires each of the clients to obtain a certificate that is linking their identity to the public key, e.g. X.509.

.NET’s authentication procedures include specific identity objects, which consist of information about the user or entity being validated. At the most basic level identity objects contain a name and an authentication type. The name can either be a user's name or the name of a Windows account, while the authentication type can be either a supported logon protocol, such as Kerberos, or a custom method designed by some application developer.

The framework also supports some additional authentication mechanisms:

- The Microsoft IIS server has specific built-in authentication mechanisms, which can be used to authenticate identities to all IIS-hosted applications. It includes all the authentication methods mentioned above.

- Passport authentication based on the participating user’s Windows account is another centralized form-based authentication service. It provides a single logon and core profile services for member sites. No further login will be necessary to access new protected resources or sites.

See Appendix C on page 104, for more information about .NET.

### 6.3 Peer-to-Peer Trusted Libraries

The Peer-to-Peer Trusted Library (PtPTL) project, developed by Intel, seeks to lower the bar to security for both small and large commercial software vendors as well as open-source developers.

It provides a high-level interface to the OpenSSL toolkit. This makes OpenSSL easy to use and provides the flexibility to support a wide variety of peer-to-peer network models.
PtPTL can, in addition to OpenSSL, also be built on top of Intel’s Common Data Security Architecture (CDSA).

PtPTL is a software toolkit that provides security components especially suited for P2P applications. It includes a wide variety of cryptographic features through object-oriented interfaces, such as digital certificates and public key encryption.

The toolkit conforms to existing cryptography standards like the widely used X.509 and most of the public key cryptography standards available. It can also use digital certificates and public key encryption to authenticate and verify the identity of a remote user over some network. PtPTL also allows other methods of peer authentication, such as authentication hardware and biometric devices to be cleanly integrated into the architecture.

It provides support for other cryptographic security mechanisms as well, such as RSA cryptography, password-based cryptography, privacy enhanced mail format, and various standard symmetric encryption algorithms.

The library also includes some support for networking and operating system primitives (like threads and locks) to allow the creation of complete secure applications.

See Appendix C on page 107, for more information about PtPTL.

6.4 Summary

The development platforms described in this chapter provide similar security mechanisms, even though their architecture and main goal differ a lot.

All platforms provide support for the most usual authentication methods, such as X.509 certificates.

It is totally up to the application developer to select the most suited platform and which of the authentication methods to apply. All of these platforms may be used to develop a new secure product.
Chapter 7

7 Conclusions and future work
This chapter will give a summary of what kind of conclusions the author has regarding P2P systems available today and about the future for P2P systems, with security and authentication methods in mind.

7.1 Conclusions
The possibilities with a peer-to-peer network are quite various. It can be used to distribute information as well as sharing resources between peers. It presents a dynamic network where information and resources can be found at several locations and are easy to require.

The peer-to-peer network architecture has very high potentials and is probably going to be more frequently used in future network applications communicating via Internet.

Security within the existing peer-to-peer solutions today is mainly based on well-tested and trusted authentication methods used in client-server environments. These methods seem to work fine also in the more dynamic peer-to-peer environment. However, some of the methods are modified more or less to suit specific P2P environments.

Authentication is just one step on the way in the creation of a secure environment, but without applying trust in the authentication solutions there will not be a secure environment.

A couple of different approaches are used in the aspect of providing trust in P2P environments. Groove allows the environmental trust to be distributed via trusted clients. Invitations to shared spaces are verified via the use of human voice-messages or via digital fingerprints. To be able to verify a digital fingerprint the client must be able to get hold of a reliable fingerprint (as reference) to see if they match. The reference fingerprint may be distributed directly from the sender or from some other trusted client (already holding the sender’s digital fingerprint). This approach will let trusted clients distribute trust within the environment.

The most used method (and one of the most trusted) is, however, the use of central authorities.

Ticket-based authentication methods, such as Kerberos, and certification-based authentication methods, such as X.509, are two examples of server-based authentication solutions frequently used in P2P environments. Authentication methods related to these techniques are also used by most
applications described in this report. These methods will provide a secure and well-trusted authentication method. They will probably also be frequently used in future P2P applications, even though they do not provide that good scalability in a system.

Identifier-based authentication methods are not usual today. There exists some example, such as Avaki’s solution (with the public key included in each object’s name). One advantage with network solutions without central authorities is that it will allow the system to become more scalable than what is possible with systems using central authorities. The main problem with identifier-based authentication is to create each object’s globally unique identifier in a secure and trustworthy manner.

Both Avaki and Groove have taken the first steps towards a dynamic peer-to-peer authentication procedure without any direct need for central authorities in their authentication processes. It seems, however, to be difficult to totally eliminate the use of central authorities. Some trusted authority is used in Avaki to provide a secure binding between the participating objects’ identities and public keys. One interesting aspect in this discussion is the new protocol (SGNP) Avaki presented (in February 2002), where each object’s global and unique identity may be generated locally (without a central authority).

Avaki’s presentation of SGNP is a draft of the protocol’s specification, but it will be very interesting to see if they really will be able to implement a reference application that provides a secure network solution without using some central authority.

Authentication methods without central authorities have been proven to be difficult (almost impossible) to implement to provide good security. This makes the solution of Avaki’s new security protocol both interesting and suspicious.

Do they really provide a secure procedure for authentication among peers from all different corners of the community? This is outside the scope of the report and the methods used in Avaki’s new protocol have to be more thoroughly investigated before some answer to this question may be found.

If some solution like the one Avaki’s new protocol permit will be proven to be secure and really scalable, similar solutions will probably be more frequently used in future P2P applications.

In case of Groove’s authentication solution, the system is designed for persons and not stand-alone machines. This will make the authentication
process easier (in smaller environments) where people may exchange invitations and digital fingerprints in person or over the phone.

To use the same solution in some environment where peers both can be persons and machines will create a problem in the way machines may get hold of authentic digital fingerprints to be able to verify the authenticity of invitations (vCards).

Access control and authorization solutions in a P2P environment is very much depending on the system itself. Most systems let local administrators (clients) select what kind of privileges different objects, or groups of objects, will have and what kind of resources and files that will be shared with others. This will work in both local and global environments.

Some local systems allow the presence of a more central administrator in aspect of setting general access rights in the system. Global solutions will, however, demand a more low-level (local) administration of access rights. It is, however, possible to combine the two solutions. An example is a company using some local P2P-related network solution administered centrally in the company with some client that need some assistance from a partner outside of the company walls. The client will be able to set local access rights for the partner, but these access rights must not exceed the rights given to the client by the company’s central administrator. This means that it only will be possible to give more restrictions on a low-level, and not possible to exceed the rights given by some more authorized administrator.

Another aspect of P2P computing is the lack of standards in the way peer-to-peer applications should communicate, which leads to an increasing amount of more or less local solutions of how communication methods will be defined. These local solutions will also prevent peers from different networks, using different P2P applications, to establish communication with each other to exchange information and resources. If some standard were available in the way peer-to-peer communications should be performed. It would probably be possible for peers running on different applications to establish connection with each other and exchange both information and resources (as the world wide web works today). This would make the dynamic network environment very large and provide the possibility for any peer to find information and resources much easier and faster than what is possible within the typical client-server environment Internet has today.
There is some work in progress with the intention to create a standard for the 3:rd generation Internet, based on peer-to-peer technology. Most of the companies behind the applications and platforms described in this report, together with other companies, are involved in this standardization project. They have at this time only had some initial meetings, so it remains a lot of work until any complete P2P standards will be presented.

### 7.2 Future work

The author of this report has looked into some of all the peer-to-peer applications available today. This is done intentionally to be able to present and focus on the differences between these P2P applications’ approach towards security and authentication methods. Some future work within this aspect may be to look into some other peer-to-peer applications available today and compare their solutions with the ones presented in this report.

Some other future work within the area of P2P systems and security may also include new peer-to-peer applications and their architectures. To see what kind of methods they will use to build secure systems. It may also be interesting to look into the way peers will authenticate each other to see if, and how, they are solving the authentication problem without involving any central authority.

Another interesting thing is to look deeper into the new protocol (SGNP) Avaki presented in February 2002. To see if (the not yet available) reference application will provide good security without using some central authority and if their authentication procedure is as reliable as it claims to be.

Last, but not least, is to keep track of the development of the 3:rd generation Internet standardizations.
Appendix
Authentication in peer-to-peer systems
Appendix A: Authentication procedures

This appendix will give a description of two authentication methods. One authentication method with the shared keys solution and one method concerning the public keys solution.

In a p2p system, a peer acts both a client and a server, which makes it well suited to explain these authentication methods in the aspect of a client-server environment. This will cover both regular client-server systems as well as peer-to-peer systems. It does, however, exist some local variants of these authentication methods in different peer-to-peer systems, but the basic functions are the same.

A.1 Kerberos
Kerberos was developed in the 1980s at MIT within a project called Athena.
It uses a shared key solution, which provides a centralised authentication server that authenticates users to servers and servers to users. This method is used instead of having authentication protocols at each server in the network.

A.1.1 Description
All participating users must be registered with the Kerberos server. This means that the Kerberos server must provide a database with the User ID and the hashed passwords of every participating user.
The server must also share a secret key with each of the participating application servers, which also have to be registered with the Kerberos server. This type of environment is called a realm.
Client and server networks under different administrations will belong to different realms. To be able to communicate between different realms Kerberos uses a realm hierarchy. This means that it will be an increased effort during authentication of users as different keys are exchanged in pairs along a connecting chain of authentication servers [L4].

Kerberos relies exclusively on conventional encryption and is built around the concept of “tickets” and central Kerberos servers. There are two kinds of servers in a Kerberos system:
- **Kerberos Authentication Server (KAS).** The KAS authenticates users at login time and issue tickets. The tickets are normally valid
for one login session and enable users to obtain other tickets from ticket granting servers.

- **Ticket Granting Server (TGS)**. TGSs issues tickets that will give users access to network service that demand authentication.

To get a Kerberos system up and running the TGSs have to receive participating user registrations, access control information, and all the necessary keys must be in place. Encryption is made with a DES-like symmetric cipher system. In which users are locally authenticated by username and password. Passwords will this way never be transmitted over the network. The participating users will only need to have knowledge of their own passwords. All other security-relevant information will be managed by a few central servers and of no concern for the user.

To be able to revoke access rights from some user the system administrator of the KAS and the TGS must update the database to make the access rights invalidated for the user. This revocation will take affect the next time the user requests a ticket from the TGS. All tickets that the user already are in possession of are, however, valid until their expire date. For example, KAS tickets usually have a lifetime of one day.

The lifetime of a ticket is a typical trade-off problem between security and convenience. If the TGS tickets have a distant expiry date the user does not have to access the TGS that often. The TGS might even be off-line for some time without too much impact on the users. This will make the access right revocation procedure to have longer delay. If the TGS issues tickets with a short expiry time, users have to update their tickets more frequently. This will make the availability of security servers to become a more important issue for good system performances.

### A.1.2 Architecture

Kerberos have all the advantages of a centralised security system. A single security policy is enforced by a limited number of security servers. It is relatively easy to check that the system setup complies with the existing security policy.

A Kerberos authentication session scenario (figure A.1) [L2]:
The session will start when a user, for example A, logs on to the local host. The user enters the username, password and requests the service of a TGS.
Figure A.1

1. **A** $\rightarrow$ **KAS** $\rightarrow$ **TGS** $\rightarrow$ **B**

   **Kerberos authentication scenario**

   1. A’s identity, the name of the TGS, the expiry date of the requested ticket and a nonce is sent to the KAS. The KAS will use this information to generate a session key ($K_{a,tgs}$) and a ticket:

   \[
   Ticket_{a,tgs} = e_{K_{tgs}}(K_{a,tgs}, A, T_1, L_1)
   \]

   2. The session key, ticket and nonce ($N_1$) are encrypted with A’s secret key ($K_a$) and returned to A. At A’s host ($K_a$) is reconstructed from the password and the session key ($K_{a,tgs}$) is obtained.

   3. A creates an authenticator ($K_{a,tgs}(A, T_3)$) and sends the authenticator, the ticket, the requested expiry date ($L_2$), a further nonce ($N_2$), and the name of the service to the TGS. The TGS decrypts the ticket using $K_{tgs}$ and verifies the ticket’s validity against the local clock, by checking that the ticket creation time ($T_1$) is within the specified time limit. Then it uses the key ($K_{a,tgs}$) from the ticket to check the authenticator. If all verifications succeeded TGS generates the session key ($K_{a,b}$) and the ticket:

   \[
   Ticket_{a,b} = e_{K_b}(K_{a,b}, A, T_2, L_2)
   \]

   4. The session key ($K_{a,b}$) and $Ticket_{a,b}$ are sent to A encrypted with the session key A shares with the TGS. A stores the encrypted ticket and decrypts the new session key ($K_{a,b}$).
5. A asks B for an authenticated session. B decrypts the received ticket, checks its validity and obtains the session key ($K_{a,b}$).

6. B decrypts the authenticator with $K_{a,b}$. After successful decryption and verification of time, B replies with the last time stamp ($T_4$) received encrypted with the session key.
   A decrypts the time stamp and compares it with its own copy of $T_4$. If they are equal B has been authenticated.

Keys and tickets are stored on each client’s machine. This makes the local security mechanisms a central part of the Kerberos total security, which means that the local protection mechanisms must be well trusted.

If users are working from simple terminals this security issue is not much of a problem. It will be much more severe if users run Kerberos from a PC or a multi-user workstation where these keys and tickets may be available for someone else using the same machine.

**A.2 X.509**

X.509, initially issued in 1988, is based on the use of public-key cryptography and digital signatures. The standard does not dictate the use of a specific algorithm but it recommends the use of RSA.

The digital signature scheme is also assumed to require a hash function. Again, the standard does not dictate a specific hash algorithm. The 1988 recommendation did, however, include the description of a recommended hash algorithm. This algorithm was later shown to be insecure and has been dropped from the 1993 recommendation.

A third version of X.509 was drafted in 1995 (figure A.5). This version includes the use of extensions within a certificate.

**A.2.1 Description**

X.509 defines a framework for the utility of providing authentication services by the certificates of its users. These certificates, generated by a CA, have the following characteristics:

- Any user with access to the public key of the CA can recover the user public key that was certified by the CA.
- No party other than the CA can modify the certificate without detection.

Because certificates are not forgeable they can be placed in a public directory without the need for any special protection.

In the case where all users subscribe to the same CA there will be a direct common trust of that CA. This means that all user certificates can be
placed in the same public directory and be accessed by all users. Another possibility is to let users transmit their certificates directly to other users. In either case, once they are in possession of each other’s certificate they will also have confidence in that messages encrypted with each other’s public keys will be secure from eavesdropping and that messages signed with their private keys are not forgeable.

If the community of participating users is large it may not be practical for all users to subscribe to the same CA. This is because each participating user must have a copy of the CA’s own public key to verify signatures. The public key must be provided to each user in an absolute secure way so that each user will have confidence in the associated certificates. In a large community it may be more practical if a number of CA’s would be present, each securely providing its public key to some fraction of the participating users.

Example of an exchange of certificates within X.509 [L3]:
Suppose that A has obtained a certificate from the CA named X1 and B has obtained a certificate from the CA named X2. If A does not securely know the public key of X2 then B’s certificate, issued by X2, is useless. A can read B’s certificate, but A cannot verify the signature. If, however, the two CAs have securely exchanged their public keys with each other the following procedure will enable A to obtain B’s public key:

1. A will obtain the certificate of X2 signed by X1 from the public directory. Because A knows X1’s public key, A can obtain X2’s public key from the certificate and verify it by the means of X1’s signature on the certificate.
2. A goes back to the directory and obtain the certificate of B signed by X2. A has now a trusted copy of X2’s public key and are able to verify the signature and may now securely obtain B’s public key.

In this example, A uses a chain of certificates to obtain B’s public key. This chain is expressed as: \( X_1 << X_2 >> X_2 << B >> \)

Using the same method, B can obtain A’s public key with the reverse chain: \( X_2 << X_1 >> X_1 << A >> \)

The current standard uses the following notation in the definition of a certificate: \( CA << A >> = CA \{ V, SN, AI, CA, T_A, A, Ap \} \)
\( CA << A >> \) = the certificate of user A issued by certification authority CA.
\( CA \{ X \} \) = the signing of X by CA. It consists of X, appended with an encrypted hash code.
X.509 includes three alternative authentication procedures. All procedures make use of public-key signatures where it is assumed that the two parties know each other’s public key. This can be done either by obtaining each other’s certificates from a public directory or by including the certificate in the initial message.

One-way authentication (figure A.2) includes a single transfer of information from one user (A) to another (B) and establishes [L3]:
1. The identity of A and that A generated the message.
2. That the message was intended for B.
3. The integrity and originality of the message.
Only the identity of the initiating entity is verified by this method.

One-way authentication messages are used simply to present credentials to B. They can also be used to send session keys to B, encrypted with B’s public key.

Figure A.2

Two-way authentication (figure A.3) extends one-way authentication and permits both parties in a communication to verify the identity of each other. This authentication procedure establishes these additional elements:
4. The identity of B and that B generated the reply message.
5. That the message was intended for A.
6. The integrity and originality of the reply.

Figure A.3
Three-way authentication (figure A.4) includes a final message from A to B. This method is needed when synchronised clocks not are available.

Figure A.4

1. $A_{\{t_A, r_A, B, sgnData, E_{K_{UB}}[K_{UB}]\}}$

2. $B_{\{t_B, r_B, A, r_A, sgnData, E_{K_{UA}}[K_{UA}]\}}$

3. $A_{\{r_3\}}$

The certificate structure and authentication protocols defined within X.509 are used in a variety of contexts. For example, the X.509 certificate format is used in S/MIME, IP Security, and SSL/TLS.

A.2.2 Architecture

Each certificate contains the public key of a user and is signed with the private key of a trusted CA.

Description of the X.509 certificate (figure A.5) [L3]:

- **Version.** The default version is 1, but if the Initiator Unique Identifier or Subject Unique Identifier is present the version must be 2. If one or more extensions are present the version must be 3.
- **Serial number.** An integer value, unique within the issuing CA, that is unambiguously associated with this certificate.
- **Signature algorithm identifier.** The algorithm used to sign the certificate, together with any associated parameters.
- **Issuer name.** A X.500 name of the CA that has created and signed the certificate.
- **Period of validity.** Consists of two dates: the first and last on which the certificate is valid.
- **Subject name.** Name of the user to whom the certificate refers.
- **Subject’s public-key information.** The public key of the subject together with an identifier of the algorithm for which this key is to be used, plus any associated parameters.
- **Issuer unique identifier.** An optional bit string field, to uniquely identify the issuing CA if the X.500 name has been reused for different entities.
• **Subject unique identifier.** An optional bit string field, to uniquely identify the subject if the X.500 name has been reused for different entities.

• **Extensions.** A set of one or more extension fields.

• **Signature.** Covers all of the other fields of the certificate, and contains the hash code of the other fields, encrypted with the CA’s private key. It also includes the signature algorithm identifier.

Figure A.5

![X.509 Certificate](image)
Each certificate includes a time period of validity, and a new certificate should be issued just before the expiration of the old one. It may, however, be desirable to revoke a certificate before it expires if:

- The user’s secret key assumes to be compromised.
- The CA no longer certifies the user’s public key.
- The CA’s certificate assumes to be compromised.

The CA must maintain a Certification Revocation List (CRL, figure A.6) consisting all revoked but not expired certificates issued by that CA, including both those issued to users and to other CAs. All CRLs must be available in the public directory to let all participating users take advantage of them. The CRLs are signed by the issuer and includes the issuer’s name and the date the list was created. They also provide the date when the next CRL is to be issued, and an entry for each revoked certificate. These entries include a serial number for each certificate as well as a revocation date. All serial numbers are unique within a CA. This means that a serial number is sufficient to identify the certificate.
When a user receives a certificate in a message, he or she, must determine whether the certificate has been revoked, or not. This means that the user must check the CRLs in the public directory each time a certificate is received.

To avoid delays and possible costs associated with directory searches, it is likely that users will maintain a local cache of certificates and lists of revoked certificates.
Appendix B: Existing P2P systems

B.1 Napster

Napster is the application that really made people open their eyes for peer-to-peer network solutions. It has been designed to let people exchange music, MP3-files, with each other in a cheap and efficient way. The main features of Napster are: file sharing, search capabilities of music files, instant messaging, chat capabilities, Hot List User Bookmark, and a user-friendly interface [N1].

B.1.1 Architecture

Napster is a hybrid peer-to-peer system with a centralized device, a server. The main purpose of this central server is to keep an index of all the Napster clients currently online. The server does not contain any MP3-files itself. They are located on each Napster participant’s local computer in a special directory that is shared within the Napster community [N2].

Computers that use modems have dynamic IP addresses and are mostly located behind a firewall. Napster solves the problem with the permanently changing IP addresses in a classic way. The users are registered using a unique nickname, thus bypassing the DNS system and permanent IP addresses. The nickname is what identifies a user in the Napster environment, and not its IP address or domain name.

Every time a client connects to the network, it will send its current IP address and port number to the server. This information is later used for direct connection between client peers. The firewall problem is solved based on the possibility of the client behind the firewall to make outgoing TCP connections, while it cannot receive any incoming messages. This is called as “pushing data”. The same method is used as well by other implementations to solve the firewall problem.

A Napster music file searching sequence (figure B.1):
1. Client opens their Napster utility, logs onto the centralized server and makes the request query active and search through the server’s file index for the requested song, or artist.
2. The server sends response of the client’s query to the client, including file names and session addresses of the computers holding the files.
3. The Client does establish direct connection with some client, holding the requested file, and starts to download the file. The connection is terminated after completed download.

Figure B.1

B.1.2 Security
The central servers store information about clients and content indexes, so there is no real anonymity and privacy in this kind of system. IP addresses are also distributed in the open, so it is possible to trace a user. The existence of a central server also makes the whole system a bit more vulnerable than a pure peer-to-peer system, in the aspect of server crashes. If a server will crash, for some reason, it also means that the network supported by that server will crash.

The authentication provided by Napster is basic. Users are identified with a nickname and authenticated with a personal password, but there is no encryption involved in the connections between client and server (so far). For further authentication will MD5, a one-way hash function, be used to send digital signatures included into protocol messages.

Another matter of security is the vulnerability in this kind of system if someone breaks into a server. They are then in such a position that they
can affect the whole network. For example, send files to any client, online, with the auto-upgrade message, which automatically will be accepted by all clients in case of a system upgrade.

The Napster website can be found at:
http://www.napster.com/

B.2 Gnutella
Gnutella does support file sharing of any kind of file types, not only music files as in the Napster environment.
It is a pure peer-to-peer application with a completely decentralized protocol, which means that there are no central servers involved. The only component part of the network, are the participating peers. The logical network is extended dynamically every time someone logs on to the network.

B.2.1 Architecture
Gnutella behaves a lot like an old fashioned game of message passing. You tell something to your neighbour, who redistributes the same message (hopefully) to their neighbour, and so on. When the last person get the message, he or she will tell the first person what the message was. Most of the times the message will change during this distribution, and everyone involved in the game will get a good laugh.
In Gnutella the messages are just passed trough every peer, until it gets a match for the request, so there are not likely any change in the request message during the distribution.

As soon as a Gnutella client comes online, it says, “Hello, I’m here” to another Gnutella client. That client tells eight other clients, which it has already established contact with, about the new client. Each of those eight clients will tell seven others, who tell six others, and so on. This way each client has a larger number of clients who know it is online.

A user never has to think about where in the Gnutella network he, or she, can find the requested files. It is enough to just have knowledge of a couple of other connected Gnutella clients to be able to, by distribution, have access to the whole network.

A typical search scenario (figure B.2):

1. User sends a request to its neighbour peers, they respond with “yes”, or “no” it they got the requested file.
2. The neighbour peers that don’t have the file will send the request to their neighbours, and so on.
3. When a peer got the requested file, it responds to the peer closest in the distribution chain, until the response gets to the request originator.
4. The user establishes a direct connection to the peer hosting the required file and starts to download the file.

Figure B.2

Every peer is both a server and a client, and all servers are equal. This means that the Gnutella network has no hierarchy. All computers connect directly to a couple of other computers. This is a very efficient way to create an enormous tree-structure in a short time period. It is possible to specify how large the network should be, by setting some specific protocol parameters, which will be used by the application in order to become a part of the network.

Peers running on very fast connections can, and will, support more traffic. They will become a kind of a hub for other peers, and in the same time get their requests answered much more quickly.

If some of a peer’s neighbours, through whom it accessed the network, disconnects from the community. The peer may see a dramatic drop of the network’s size. This does not mean that all of the other peers on the network went offline, just that the peer’s connection to the whole network no longer is complete. There exists some “oases” of sub-networks and they will connect to each other, and break apart, from time to time,
depending on how peers will go on-line and off-line. One minute it might be possible to see 10000 hosts in the network, and the next minute it can drop down to just 1000, but most of the hosts are still online, on another part of the network.

### B.2.2 Security

Pure peer-to-peer networks, like Gnutella, with this kind of decentralized approach are virtually impossible to shut down. There are no easy ways to target and stop the use of these applications since there exist no central server, which are maintaining indexes of the participating clients.

There are no usernames, other identities, or passwords travelling around the Gnutella network. It is not much information in a query to the network that can link queries back to the originator. It is, however, not totally impossible to figure out whom is searching for what, but it is unlikely. Each time the query is passed through some peer the possibility of discovering the originator is reduced exponentially. In short, there is no safer way to search without being watched. Not even the originator’s closest neighbours know that it was he, or she, that sent the request, just that the request was linked through their machine. The answer to the request will be linked back to the originator, but he or she doesn’t have to admit that they are the originator.

Downloads, however, are not anonymous for performance reasons. It is a lot quicker to download files directly between two machines, instead of downloading files via a distributed network.

Communications between peers are not encrypted, but there are some client applications trying to use encrypted communication with some kind of centralized unit to validate the certificates.

As in the Napster case, a peer cannot connect directly to some other peer behind a firewall. The peer, behind the firewall, must push the request out through the firewall, to establish a connection. But if both peers are located behind firewalls, they cannot establish a connection between them. No peer is allowed to connect to the other behind the firewall. It will require a third party, which is able to link the two peers together, and this third party must not be located behind a firewall.

The Gnutella website can be found at:

http://www.gnutella.com/
B.3 Freenet

Freenet is a pure peer-to-peer network with a decentralized architecture, without a central server. It was originally designed by Ian Clarke, but is now being implemented on the open-source model. It has been designed to transparently store, move, replicate and delete files according to certain policies, providing efficient services without using central location indexes as in the Napster environment, and without broadcasting messages as in the Gnutella environment.

The major tasks of a Freenet network is not only file sharing, it is mainly about disk space sharing. This disk space sharing is used for data management according to the established network policies.

Freenets definition of itself is that: “Freenet is a peer-to-peer network designed to allow the distribution of information over the Internet in an efficient manner, without fear of censorship. Freenet is completely decentralized, meaning that there is no person, computer, or organisation in control of Freenet or essential to its operation”.

B.3.1 Architecture

Each peer shares a part of its disk space, which is available for reading and writing to all the community. Each file in the data store is associated with a location independent key and an address of the node where that file came from, plus eventually some metadata about the file. The routing table of each node contains the addresses of other nodes and the keys they hold. It may also include some keys for deleted files together with the address to a node that may still have the file. The files are moved around the network, and will be placed at locations where there is more interest in them, and deleted from locations where there is no interest in the files. The users do not even know what files they have stored locally. This file management makes it almost impossible to know where the files come from and who is the originator. Every peer sees each request as if it was originated at one of the nearest neighbour peer.

A typical search scenario (figure B.3):

1. A node sends a data request to its neighbor.
2. Node does not have the requested data, so it is checking its routing table, and sends the request to the selected neighbor node.
3. Node does not have the requested data, and no more neighbors online, so the node replies with a “failure” message.
4. Node checks its routing table again and selects another neighbor to send the request to.
5. Node does not have the requested data, so it is checking its routing table, and sending the request to the selected neighbor node.
6. Node does not have the requested data, so it is checking its routing table, and sending the request to the selected neighbor node.
7. Node already has seen this request, so it will reply with a “failure” message to avoid generating any loops.
8. Node does not have any more neighbors online, so it replies with a “failure” message.
9. Node checks its routing table again, and selects another neighbor to send the request to.
10. Node got the requested data, so it replies with a “success” message and sends the requested file to its upstream neighbor.
11. Node will cache the file in its data store and update its route table. It will then send the data reply to its upstream neighbor.
12. Data response reaches the request originator and the requested data can be presented to the user.

Figure B.3

The architecture of Freenet is similar to the Gnutella architecture, but the high-level protocol is completely different. The strength put on anonymity and privacy has also made the high-level protocol quite complicated. Connected peers that communicate with each other, using messages, manage the Freenet’s network. These messages actually form the high-level protocol layer. There exist also a low-level transport protocol, including standard protocols such as TCP and UDP.
There are five main design goals in Freenet development:

1. Anonymity for both producers and consumers of information.
2. Deniability for peers storing specific information within Freenet.
3. Resistance to attempts by third parties to deny access to information.
4. Efficient dynamic storage and routing of information.
5. Decentralization of all network functions.

**B.3.2 Security**

Communication and data are encrypted and digitally signed, so each user is not sure about what is stored on the hard disk. Only the original creator has the possibility to modify the files content.

The current key exchange system is Diffie-Hellman. Key formats are hashing descriptive text strings, resulting in a 160-bit SHA-1 hash code. Here is an example of how this key exchange works:

First two nodes, A and B, agree on a large prime \( n \) and \( g \) such that \( g \) is primitive mod \( n \). These two integers do not have to be secret; A and B can agree to them over some insecure channel. They can even be common among a group of users.

Then, the protocol goes as follows:

- A chooses a random large integer \( x \) and sends B: \( X = g^x \mod n \)
- B chooses a random large integer \( y \) and sends A: \( Y = g^y \mod n \)
- A computes: \( k = Y^x \mod n \)
- B computes: \( k' = X^y \mod n \)

Both \( k \) and \( k' \) are equal to \( g^{xy} \mod n \). No one listening on the channel can compute that value; they only know \( n \), \( g \), \( X \) and \( Y \), unless they can compute the discrete logarithm and recover \( x \) or \( y \), they do not solve the problem. So, \( k \) is the secret key that both A and B computed independently.

Keys provide a way to stop cancer nodes responding with bad data. For this to work every node must check the document it gets. Chains of tunneling nodes need some way to signal that one has found that the tunnel is bad. If they simply passed the data on they would be seen as the cancer node.

The Freenet website can be found at:

[http://freenetproject.org/](http://freenetproject.org/)
B.4 Mangomind

Mangomind is a Microsoft Windows file system based on Mangosoft’s patented peer-to-peer pooling technology. It is an Internet file sharing service developed by Mangosoft Inc. The application allows multiple users to simultaneously access, share, or store files from all over the world in the exact same manner as if they were on their local PC.

It is integrated in the Windows system and looks like a shared network drive in the regular Windows Explorer window. This makes it possible for all Windows applications to work with the Mangomind’s services without any special configurations. It also makes it quite easy for the users to access already uploaded files, and to add new files into the Mangomind drive area.

Users have the possibility to work with the same Mangomind drive from different PCs. The user can use a portable identity file to verify its identity.

The portable identity file contains everything the Mangomind system requires to uniquely identify a user. This file can also be used to recover a disk connection in case of a local computer crash, where all information about the Mangomind drives has disappeared. If the user has more than one Mangomind drive connection, he or she should make a portable identity file for each of those drives, to ensure future access to these drives.

It is highly recommended to store these portable identity files on a floppy disk in a secure place, and not only on the computer’s hard disk in case of a local disk crash.

All the Mangomind drive files are backed up every day onto tapes on the server side, to ensure no loss of data in case of a server crash. This utility also allows the possibility to recover files deleted by mistake. The Mangomind Resource Manager can restore such files from the most recent backup tape.

B.4.1 Architecture

The central part of this file sharing solution (figure B.4) is the central Mangomind server, where the Mangomind drives are located. These drives are just a storage space available to the initiator and all the users he, or she, invite to the shared drive.
Every stored file is protected by a 128-bit Mangosoft-patented implementation of the BSAFE Crypto-C encryption from RSA Security, which is standard in the Mangomind drive to ensure privacy. The encryption is done on the clients local machine just before the file is transferred to the Mangomind drive. This will make the files unreadable for both Mangosoft and the hosting service provider. It is only the authorized users that can decrypt and read the content of the separate files stored on the Mangomind drive. Drive administrators do have full access to all the files and folders on a Mangomind drive, to be able to solve any problem with file properties. This could be if users have left the Mangomind drive without leaving any user with access rights [M2].

When a new drive is created at a Mangosoft server, the first user (the initiator) will automatically become administrator for that drive and be given special privileges.

The initial representation of a user is created from their email address. It is used to include a unique join link in an invitation message, containing a password that belongs to that specific user. This password can only be used once. After the user’s first connection (via HTTPS) to the Mangomind drive the password is invalidated. This means that no person...
can impersonate the initiated user later and get access to the Mangomind drive.
The drive administrator has the possibility to initiate invitations to users via email, and place them in user groups, or as individual users on the Mangomind drive. All participating users must be given an invitation from the administrator, to be able to access the drive. The administrator also has the possibility to give specified privileges to each user, or group of users, to suit the system requirements. It can also delegate administrator privileges to some other invited user [M1].

After a user has been invited to a drive and verified to have a valid invitation-password (by the server), the client’s Mangomind application generates a private/public key-pair and sends the public key to the server for further authentication. The drive and its stored files are only accessible to clients authenticated by the Mangomind application using the client’s public/private keys. The authentication is performed automatically each time a user connects to the Mangomind drive. The host uniquely identifies each user based on their Mangomind username and the identifying Mangomind information stored on their local machine [M1]. The files are, however, not encrypted when they are stored in the user’s local cache, which improves the access time when the user is working with the files locally.

The authentication process is using mutual authentication, where both the server and the client authenticates each other. The process starts when the client sends a random string, signed with its private key, to the server. The server verifies the message, using the client’s public key. The server signs the message with its private key, and sends it back to the client. The client verifies the message, using the server’s public key. If the result is equal to the client’s original random string, the client knows that the server is who it claims to be. The same procedure is performed by the server, to be able authenticate the client [M3].

The Mangomind website can be found at:
http://www.mangosoft.com/products/mangomind/

**B.5 WebRiposte**

WebRiposte is a web-enabled peer-to-peer middleware messaging and data management solution, developed by The Escher Group. It is developed to provide a solid base for e-business applications.
WebRiposte provides a secure foundation upon which distributed applications can be built and deployed. It supports secure, web-based transactions and application development platform. This means that developers only need to focus on the application layer to be able to produce reliable and scalable applications.

It is not required that a user must have a constant on-line connection. Users are allowed to work off-line and the data content are synchronized with the server when the user reconnects to a Message Server.

**B.5.1 Architecture**

The WebRiposte Proxy Server provides secure, efficient access to third-party web content. It stores web content in the Message Store, and efficiently distributes this via the messaging system, by maintaining a dual-level distributed store of cached web content.

It uses a WebRiposte Proxy Agent to service remote client requests for content, either from the cache or from the origin server. The Proxy Agent runs as an online service on the Correspondence Server and forward requests for content that has not been cached to the origin server. It also distributes the response to the requesting client and updates the cache. The agent also re-verifies the content freshness for any content being served from the cache.

The WebRiposte HTTP Server provides a web-based interface to the Riposte Message Server, which is a central part of the system. It handles requests from XML and web browser clients. The HTTP Server also, transparently, serves content stored in the Message Server to a web browser via the standard HTTP interface.

Static content may be published at the Correspondence Server and distributed via specific subscription groups. It may also be dynamically requested via the WebRiposte Proxy Server, and then cached for subsequent use by the HTTP Server.

The system supports XML Web Services, such as SOAP and WSDL, and Microsoft’s .NET framework, which will make the system very flexible and dynamic. The HTTP Server will serve requests from XML-based client applications that issue API calls via the SOAP programming interface to the Message Server.
The HTTP Server also supports HTTPS, which provides secure connections using the SSL/TLS protocol [W1].

This is a view of how a WebRiposte network may look:

Figure B.5

A username (an identifier) is constructed as a string, which includes the following fields: \([\text{site-ID}/ \text{group-ID}/ \text{node-ID}]\)

This identifier is bounded to an IP address, which is known (local, private) or can be found via some DNS.

B.5.2 Security

The infrastructure provides a complete messaging system solution, providing both message distribution and message storage, as well as, providing a secure, high performance content delivery platform.

WebRiposte supports secure messaging, using SSL/SET/TLS protocols. These mechanisms secure both the communication between peers and provide secure communication to applications via HTTPS. SSL will provide security by ensuring that unauthorized third-party users cannot intercept sensitive data.
User authentication and management of local security policy is based on close integration with the Windows NT/2000 security model and data encryption is supported via integration with the Windows CryptoAPI [L1]. The system will use data encryption and authentication to secure transmission of data throughout the system, and digital signatures to prevent any data tampering. The application will automatically verify the digital signatures when data content is retrieved from a server. This does not only apply to software components, but to HTTP pages as well. WebRiposte uses these secure protocols to support:

- Secure replication of messages between Correspondence Servers and remote Message Servers.
- Neighbour authentication by Correspondence Servers and by clients.
- Secure content delivery to all web clients.

WebRiposte also supports the definition of ACLs on objects, which can be used in the process of permitting or denying access to web content stored in the Message Store, for specific users and user groups. In addition, the Web Proxy Server supports Internet security by letting system administrators restrict user access to external web sites, through the use of access lists that define allowed and disallowed site addresses.

The security policy is replicated among all peers to ensure consistent enforcement within every peer group. All certificates are managed centrally and distributed to peers via replication. WebRiposte does, however, not include any built-in encryption technology. It is left to the customers to integrate encryption, based on local requirements.

The WebRiposte website can be found at: http://www.eschergroup.com/prod-tech-comp-webriposte.html

**B.6 Groove**

Groove support direct connections between nodes and is developed by Groove Networks Inc. It provides a platform well suited for the development of peer-to-peer collaboration applications and offers a basic toolset that includes file sharing, instant messaging, calendaring, and co-browsing, among other functions. It also provides conferencing capabilities, including voice options.

The fundamental idea of Groove is the shared space. It is private and viewable only to those who have agreed to participate in the shared space.
The participant users of a shared space can collaborate with each other via a number of predefined tools and applications, or via an unlimited set of applications created by different developers.

It provides users with much control over the functionality of a space itself. If any participating member decides the shared space needs new or additional functionality, he or she can add new components or tools on the fly.

When some user adds a tool, the system sends a command to all other users to do the same. If the other members already have the tool installed, Groove automatically adds the tool to the shared space. But if any member does not have the tool installed yet, the application locates and downloads the tool from any recognized Groove Component Server. The application will install the tool locally and add it to the shared space.

The system promotes the use of different “personas”. This means that a user can have several identities, as a “work” identity, an “online game” identity, and a “family” identity all in one place. Technically, a single account in Groove supports the use of multiple user names [L1].

Each user in a shared space is allowed to collaborate with all other peers within the shared space by exchanging digital signatures. It is possibly in this kind of strongly encrypted peer systems to create an easy-to-use VPN across the Internet, where the public key infrastructure is obviated by the presence of exchangeable identity markers on each client.

**B.6.1 Architecture**

Groove software uses a direct peer-to-peer communication by default. When a firewall makes direct interaction impossible Groove software automatically sends the traffic to a centralized relay service, where it is encapsulated as HTTP and “tunneled” through an available port.

The Groove platform takes advantage of integration tools – client-side agents and server-based “bots”. The bots provide support for storage of customer history, transaction processing, records/document management, and allow developers to automate processes associated with data residing in centralized systems.

The part of Groove’s architecture that is visible to the end users is called the transceiver (figure B.6). It contains the user’s variety of shared spaces.
with differing sets of members and the collection of tools available within each shared space.

Figure B.6

<table>
<thead>
<tr>
<th>Transceiver</th>
<th>Shared Space</th>
<th>Shared space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool</td>
<td>Tool</td>
<td>Tool</td>
</tr>
<tr>
<td>Security</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XML Object Store</td>
<td>Connection</td>
<td>Synchronization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>XML Object Routing</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Awareness</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Transparency</td>
</tr>
</tbody>
</table>

The underlying peer service, include security, local XML object storage, and XML object routing services [GR1]:

- **Security Services.** Groove handles user public/private key creation, distribution and maintenance automatically. It also handles end-user authentication, and 192-bit encryption of all content and activity stored locally on disk and travelling across a network. Groove’s security is protected against end user indifference and imposes no additional cost associated with centralized administration and management of certificates or encryption keys.

- **Storage Services.** In Groove are all contents, gestures and activities captured for local storage in the XML object store.

- **Synchronization Services.** All members in a shared space maintain a local copy of the space. Groove’s synchronization services keeps the contents of each copy completely synchronized with all other members’ copies.

- **Peer Connection Services.** Groove will automatically and transparently locate users with dynamically assigned IP-addresses. It traverses both secure firewalls and NATs.

**B.6.2 Security**

Groove encrypts all content on disks as well as the content being transmitted over the network. Groove also automatically encrypts the user’s account, all shared spaces, and the local content on the user’s device.
Users must create a passphrase to encrypt their account and shared spaces, which makes it more difficult for someone with malicious intent to make an educated guess at a user’s secret code. If an intruder will gain access to the file system in which Groove spaces are listed, the contents of those spaces will remain encrypted and impossible to compromise without the user’s passphrase.

The security design set out to reconcile two opposing factors [L1]:
- To provide a complete end-user control.
- The need for security in and around the content and activities of a shared space.

In more detail, the Groove security model address these challenges:
- Protect the confidentiality of content and activity in a shared space.
- Allow users to authenticate the identity of other users.
- Lock the shared space from access by unauthorized users.
- Protect the shared space from inadvertent or malicious spread of viruses.

In this kind of peer-to-peer architecture, all security function and management is relegated directly to the peer device themselves. This means that it is no server-based administration of security. To get this approach effective the peer-computing environment must not only provide robust security, but also do so in a way that is effectively transparent to the users.

Groove approaches this problem by letting users establish initial trusted relationships via the exchange of invitation cards in the form of “vCards”. The vCard includes information about a peer, like the peer’s identity and its public key.

When a peer accepts an invitation, often sent via email, it automatically also has accepted the sender’s vCard and public key. The receiving peer must reply by sending its vCard in return, which is done automatically within Groove.

Two main methods are available to users to ensure that an invitation does indeed come from the user who sent it. This is important, since an invitation sent e.g. via email depends on how much the user will trust the email security system itself.

The two methods are:
- **Digital fingerprints.** When a peer sends the first message to some other peer, the digital fingerprint of the sender will be computed by the Groove application on the recipient’s local machine by using the sender’s included public key. The user may verify the sender’s
fingerprint in one of the following ways: in person or over the phone with the sender, or via some other trusted media or user (already in possession of the sender’s fingerprint). If the digital fingerprints match, the receiver will know that the sender is who it claims to be and may store the fingerprint as part of the sender’s vCard in the local account. The stored digital vCard may be used to verify subsequent invitations from the sender.

This kind of distribution of public keys ensures security without requiring any centralized certificate and key management servers.

- **Voice annotation.** Groove uses a possible approach that emulates the way people authenticate one another in real life. This approach uses all human forms of recognition, as opposed to the typical methods used by public key infrastructure-based systems. In which the peer always trust that someone “notarises” that it is who it claim to be.

  When a user receives an invitation from some other user, he or she, authenticates the user via an included voice message. To take this authentication method one step further, it is possible to add a “personal shared secret”.

All other communication is performed with signed messages, which will be authenticated by the recipients using the corresponding public key.

Two asymmetric keys are related to an identity. One asymmetric key-pair is used for signature and verification and the second asymmetric key-pair for encryption and decryption. They are generated locally without any central authority, by the Groove environment [GR2].

There are a couple of different algorithms used in a Groove system, including [GR3]:

- **MARC4** – secret-key algorithm, used for shared spaces.
- **ElGamal** – public-key algorithm, used for encryption of personal messages.
- **RSA** – public-key algorithm, used for signature of messages outside a shared space.
- **ESIGN** – public-key algorithm, used for signature of messages within a shared space.

Only invited members can participate in a Groove shared space. After a user has accepted an invitation and the users have authenticated each other, Groove will send a set of keys to the user together with the shared
space itself. These keys will allow the user to decrypt the shared space’s content. This exchange of keys for the space itself provides an easy mechanism for managing membership for the duration of a space. Throughout the normal lifecycle of a shared space, it is likely that new members will be invited and some old members will have their invitations rescinded. This presents a security challenge, since the ejected member still retains the keys to the shared space. Whenever a member is evicted from a shared space Groove automatically will issue a new shared-space key to all members so that subsequent data is protected and kept private from all past members. As far as the evicted member is concerned all “old” shared space is still on their local machine. They can still review the content and history of that space. However, they can no longer look at the new content and activities.

The Groove website can be found at: http://www.groove.net/

**B.7 Magi Enterprise**

Magi Enterprise, developed by the Endeavors Technology, gives peer status to PCs, laptops and Win CE handheld devices. It allows all participants to securely communicate and interact with each other. Magi focus on the sharing of information and acting on it anywhere and at any time. To achieve this goal Magi also provides chat and messaging capabilities on any Internet-enabled device. Magi provide support for multi-way communications between servers, PCs, PDAs, Internet-ready phones, and embedded chips.

The user identity is a pair of character string separated by an apostrophe. The first string is the user name, and the second string represents a location. For example, “Alice’Office”, “Bob’Laptop”, or “Carol’Palm”, are all legitimate Magi names. A buddy list contains all peers a Magi user can establish a connection to [L1].

**B.7.1 Architecture**

When a peer starts a session, it connects to a dynamic DNS, which is a specialized peer that uses a SQL database to manage the buddy list. It is also used to provide authentication of peers, provide directory service and support of store-and-forward capabilities.

Magi require that a firewall permit standard outgoing HTTP connections (perhaps through an HTTP proxy) to support communication without
Authentication in peer-to-peer systems

involving a third party. But if both peers are behind separate firewalls, communication is further restricted. Since neither user may be reached from the outside, a relay service must be set up to store events from one peer and forward them to the other peer when it makes a request. The relay service represents another specialization of a Magi peer to support the Magi network.

The possibility to obtain all functions of Magi in a NAT configuration depends on the ability of the address translator to map a unique identifier either automatically or through configuration options. This can be done based on either a specific port or specific URL assigned by the NAT to the machine on which the Magi peer is running.

Figure B.7

The Magi architecture (figure B.7) includes some important parts [MA1]:

- **Request Manager.** The request manager receives the parsed HTTP request from the underlying server and invokes services within the peer to handle the request. Once the request manager passes a
request to a suited module, the module can provide a response to be returned, or pass the request along to another module for additional processing.

- **Event Service.** The event service receives the events, packaged in an HTTP request from the request manager, and invokes the services that have registered interest in those event types. In addition the event service provides mechanisms to allow components to send events to other peers.

- **Buddy Manager.** Buddy names form the basis for representing the identity of individual Magi users and controlling the access relationships between Magi peers.

- **Access Controller.** The access controller uses notions of buddies, devices, and groups to control access to resources on a Magi peer. Information provided in the incoming event or HTTP request serves to identify users as being permitted defined levels of access.

- **Module Container.** It is possible to extend Magi’s functionality by dynamically loaded extension modules that provide additional functionality for the applications. The modules can be incorporated dynamically to allow custom configuration of Magi peers, in some cases even while the peer is running. Once loaded, these extension modules are provided access to the Magi APIs. They can be mapped to handle incoming HTTP requests, register events, send events to other Magi peers, and access information about buddies and access controls.

### B.7.2 Security

Security between Magi peers employs a combination of X.509 PKI governed by a Magi CA, HTTP content encryption, SSL, and ACLs. This combination allows security certificates from recognized X.509 CAs to establish the true identity of any Magi-enabled peer device. This ensures that peers communicate only with known authorized peers and that random hosts outside of the protected network cannot be used as launching points for an attack.

The application utilizes HTTP, WebDAV and other open standard protocols to create a secure cross-platform environment for collaborative-intensive applications. Where it securely links office and project teams together for such collaborative needs as file sharing, file searching, instant messaging and chat. Magi Enterprise is also suited to use to establish VPNs between peers.
Authentication in peer-to-peer systems

Each Magi peer will generate its own RSA key pair during initialisation and send the public key along with a username and password to the Magi CA. A password is used as verification for who may apply as well as subsequent authentication in case of a compromised key. If the password is accepted the CA binds the username to the public key with a digital certificate. In this way a given deployment of Magi peers may be configured to only recognize other peers of the given Magi CA’s choosing. This defines a unique name space that can not only use standard human readable names, but also correspond to an existing name space. If some peer becomes compromised the Magi CA issues periodic CRLs.

ACLs controls which users that will have privilege to use which resources. All peers maintain an access control list stating what resources they make available, who is allowed to execute them, and what requirements in terms of authentication and encryption. This ACL is remotely configurable in controlled environments by the Magi CA. In this way a central authority can make sure that the peers in its domain are not exposing any sensitive material to the wrong peers. In an open environment this is left entirely up to each end user.

Every HTTP request received by any Magi peer passes through three security checkpoints, authentication, authorization, and encryption, which are managed by the Access Controller component.

Magi permit support for four standard authentication policies common to the Web and the Internet [MA1]:

1. **Basic.** The policy for Basic authentication is defined in the HTTP 1.1 standard. Every request is accompanied by a username and password. Basic authentication has the advantage to be supported by all web browsers. This method does, however, provide a rather weak form of authentication, making it suitable only in situations where no other mechanism can be used.

2. **MD5.** The policy for MD5 authentication is defined as a part of the WebDAV standard. It provides a more secure challenge/response mechanism than the Basic method, and makes it significantly more difficult for a third party to intercept and decode a user’s name and password.

3. **Tokens.** One of the most commonly used authentication mechanism in Magi is a Kerberos derived security system called “Token Authentication”. It is a centrally managed authentication mechanism that authenticates “buddies” using a MDNS.
When Magi peers have been authenticated by the MDNS, they must obtain a session key from the MDNS to be able to establish direct communication with each other. The session key is then shared between the Magi peers and the MDNS for the duration of the session.

One of the biggest advantages with this method, like with Kerberos, is that access is centrally maintained. This is especially attractive for environments where access and resource control change frequently and must be enforced instantaneously. It also shares a major disadvantage with Kerberos. Centrally maintained access mechanisms implies that systems will have difficulty scaling beyond workgroups and that dependencies on the central authority will inhibit some of the natural advantages of peer computing.

4. **SSL Certificates.** SSL certificates are defined as a part of the HTTPS standard. The MDNS acts as a trusted third-party and CA, issuing a unique tamperproof certificate to each peer. A Magi peer will send its unique certificate to issue a request to another Magi peer. The receiving peer will verify the authenticity of the certificate, and hence the identity of the peer using the MDNS service. To improve efficiency, Magi peers will cache certificates in a private key ring to avoid making repeated requests to the MDNS service. A private key ring can also be consulted when the MDNS is unreachable, such as when two peers are using a private network without access to the Internet.

SSL certificates seem to be increasing in popularity by Magi users, but many users do also use Token Authentication. These are the two main authentication methods used by Magi peers.

Both the Basic and MD5 policies are fast but prone to password discovery and therefore not recommended for use where secure transactions are required. These methods are used to allow non-buddies to connect to a Magi peer.

Access to each and every Magi resource can be controlled with ACLs. In Magi, the URL used to refer to a resource serves as the namespace used to govern its access. Magi’s Access XML component maps these namespaces to a corresponding set of permissions using a hierarchical tree structure represented in XML. Every Magi URL is represented as a node in this tree and each node is annotated with a description of the resource, its attributes, and the authorization policy used to govern access to it. All
the components within a Magi peer can query the Access XML component. After a Magi peer determines who sent it a request, it queries the Access XML component to determine whether the sender has the necessary permissions to access the given resource. The Access XML component considers the identity of the sender, the URL of the resource, and the HTTP method requested before returning a response. The Access XML component consults each of the loaded security modules in succession. Permission is granted if any one of the loaded security modules authorizes the request, otherwise permission is denied. If permission is granted the request is processed and handled by the particular Magi module responsible for the namespace identified by the URL.

Magi are using the SSL technique for communication encryption. Which is the standard for secure Internet communication, trusted by financial and government institutions worldwide, and supported by most popular Internet Web browsers. The secure communication between peers does some times not need as heavy weight protocol as the SSL. In these cases, Magi use an SSL-like encryption method they call HTTP Content Encryption. With this encryption method the symmetric key is negotiated between the two peers using a specific symmetric key negotiation protocol. By negotiating this unique symmetric key for each message the source of the message is non-refutable and the message can be tested against being replayed. Magi also include a complete SSL implementation, which utilizes their existing PKI as well as the strongest encryption level allowable for a given deployment.

The Magi Enterprise website can be found at: http://www.endtech.com/enterprise.html

B.8 Avaki

The goal of Avaki, developed by the Avaki Corporation, is to enable all forms of P2P computing. It is an integrated platform that runs on all major operating systems. The vision for the product is to create the illusion and the benefit of a single virtual machine that encompasses devices from supercomputers to handheld devices, crosses organizations and geographical boundaries, and incorporates diverse architectures and operating systems.
Avaki is comprehensive and innovative grid software that enables secure sharing of data, applications, and computing power within the environment. It enables the possibility to collect resources, computing power, data files, and applications, to be used as a single virtual operating environment. Avaki can also streamline system management and security across locations, platforms, and administrative domains.

With Avaki there is little or no real need for central administration of the grid. Resource owners are administrators for their own resources and can define who has access to them and their resources. Initially administrators cooperate in order to create the grid. After the initialisation, it is a simple matter of which management controls the organization wants to put in place.

Avaki grids can operate across firewalls, NATs, WANs, and VPNs, and function with high-bandwidth or low-bandwidth connections. Avaki can extend across multiple clusters as well, creating a “grid of grids” where resources are shared locally and across locations. To ensure secure access, predefined policies govern where applications may be executed or which applications can be run on which data files. Avaki matches applications with queues and computing resources in a few different ways [A1]:

- Through access controls. A user or application may or may not have access to a specific resource.
- Through matching of application requirements and host characteristics. For example, an application may need to be run on a specific operating system.
- Through prioritisation, based on policies and load conditions.

Avaki will move data files, applications, or find processing power as needed, as long as the resources have been made available to the Avaki grid and the policies allow them to be used. The work of system administrators is also simplified, because they can manage the grid as a single system.

The Avaki protocol enables location transparency. Any given Web Service does not need to know the physical address of other connected Web Services. This means that Web Services can move from node to node and have multiple replicas across nodes. This is important to be able to handle automatic recovery from node failures, i.e. by transparently moving the Web Service to another node. It is also important in the
manner of enabling performance management, i.e. by instantiating multiple replica copies of a Web Service on less loaded nodes.

**B.8.1 Architecture**

The foundation for the Avaki grid software is a high-level grid protocol that is adapted to run on top of the XML and SOAP Web Services standard. This is similar to the manner in which Avaki integrates with the Sun JXTA protocol.

In Avaki is the middleware platform (figure B.8) is organized in three layers:

- The Core Services layer
- The System Management Services layer
- The Application Services layer

![Avaki's distributed middleware architecture](image)

Every Avaki object is assigned a unique and immutable identifier upon creation. The basic identifier data structure (figure B.9) consists of a sequence of variable length binary string fields. The first three play a key role in the identifier-to-object address binding mechanism [L1]:

- **Domain identifier.** The first field is used in the dynamic connection of separate Avaki systems.
Authentication in peer-to-peer systems

- **Class identifier.** It is a bit string uniquely identifying the object’s class within its domain.
- **Instance number.** It distinguishes the object from other instances of its class.
- **Security information.** It may contain a public key for encrypted communication with the named object.

Figure B.9

<table>
<thead>
<tr>
<th>Domain Identifier</th>
<th>Object Class</th>
<th>Instance Number</th>
<th>Security Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avaki’s Identifier Object</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

B.8.2 Security

The goal of Avaki’s security approach is to eliminate the need for any other software-based security controls, substantially reducing the overhead of sharing resources. With Avaki security in place users need only to know their sign-on protocol and the Avaki pathname where their files are located.

A safe sharing mechanism relies on highly granular access control, low-overhead authentication, and ironclad protection of data in transit. Robust built-in security is the keystone of Avaki’s approach to resource sharing.

Avaki’s approach to authentication is lateral and decentralized, not requiring appeal to a central trusted authority, even though such an authority is useful to establish the initial peer and resource identities. Authentication is based on Avaki’s approach to resource identities. Each resource made available to the Avaki grid has a unique identifier that is related to, but independent of its user-visible name.

As users access files run applications or submit jobs to queues, Avaki must authenticate the resources and ensure that the requested processing is, in fact, allowed. Avaki’s approach to authentication is lateral and decentralized. This approach enables cross-company collaboration in situations where the different companies have completely different security technologies.

The unique identifier is also independent of its physical location, resulting in some significant advantages [A1]:

- Avaki can move a resource physically without affecting its identity or name.
Avaki can create multiple copies of the same resource without affecting its identity or name.

The unique identifier is used to authenticate resources. Avaki incorporates a PKI-based public key identifier through which each resource is authenticated automatically. As a result of this last feature a resource can authenticate another resource without appealing to a trusted certificate authority. Avaki’s approach to authentication based on PKI results in less overhead, greater privacy, and lower risk of failure. It also provides high scalability [A2].

Every Avaki object has a public key pair. This public key is part of the objects name. Objects can use this public key of a target object to encrypt their communications. Likewise, an object’s private key can be used to sign messages, providing authentication and non-repudiation. The integration of public keys into object names allows Avaki to avoid the need for a certification authority (although such an authority is still useful for establishing use identities). If an intruder tries to tamper with the public key of a known object it will only create a new name, which is unknown to the community [L1].

Going beyond the access control capabilities of standard file systems, Avaki offers an allow-all-but/deny-all-but protocol that provides maximum granularity and flexibility. Access to individual resources, such as files, directories, applications, and processors, can be controlled separately with users and groups being granted or denied access.

Figure B.10

Avaki’s security model (figure B.10) has two layers:
- The message layer is responsible for ensuring that communications between Avaki objects are secure.
• The access control layer is responsible for access control and it determines what objects are allowed to call a particular object’s methods. It also relies on the message layer for some services, such as encrypting rights certificates.

B.8.3 Secure Grid Network Protocol
Avaki has recently (in February 2002) presented a draft of a new security protocol. It is called the Secure Grid Naming Protocol (SGNP). The protocol does not introduce the need for central trusted authority, thus enabling the creation of scalable Grid systems and facilitating different models for Grid administration.

SGNP defines a scheme for location-independent logical naming of grid resources, in which the identities does not necessarily have to depend on some CA or trusted third parties. The globally unique object identities may instead be generated locally.

The different fields in a SGNP Location-independent Object Identifier (LOID, figure B.11) are [A3]:

• **LOID Type** – It is unique for each type of LOID.
• **Domain Resolver ID** – It identifies the Domain Resolver, which has ownership of the Grid Resource.
• **Binding Resolver ID** – It identifies the Binding Resolver within a domain.
• **Object ID** – It identifies a Grid Resource Binding within a Binding Resolver.
• **Security Information** – It contains the security information for a Grid Resource.

Figure B.11

<table>
<thead>
<tr>
<th>LOID Type</th>
<th>Domain Resolver ID</th>
<th>Binding Resolver ID</th>
<th>Object ID</th>
<th>Security Information</th>
</tr>
</thead>
</table>

The SGNP Identifier Object

SGNP uses the same kind of naming structure as the current Avaki protocol. Both protocols use some kind of security information included in the object’s names. This security information can be:

• Nothing. To provide compatibility support for current naming systems and other systems.
• RSA Public Key.
• X.509 certificate.
• OpenPGP certificate.
In the case of a RSA Public Key in the Security Information field, a peer client may establish mutually authenticated communication with some resource (or client) if the client trusts from whom it received the identity of the other resource (or client). All messages will be signed with the sender’s private key and encoded with the recipient’s public key. This will allow each party to verify the sender/receiver of each message.

A Grid Naming Service (GNS) and a Resolver hierarchy (figure B.12) build up the SGNP. All clients and resources will call a GNS (which has a well known binding) when they would like to find some other resource (or client), in the environment. The GNS will provide clients and resources access to the SGNP naming service, where a Resolver hierarchy maintains the authoritative mapping between LOID’s and bindings. The GNS will, by traversing the Resolver hierarchy, obtain the binding to the requested resource (or client) and return it to the requesting resource.

Figure B.12

A description of the Resolver hierarchy (figure B.12) [A3]:

- **LOIDResolver.** It maintains the authoritative mappings from DomainResolverID to DomainResolver bindings.
- **DomainResolver.** It maintains the mapping of BindingResolverID to BindingResolver bindings.
- **BindingResolver.** It maintains the authoritative mappings from ObjectID to resource bindings.

To improve performance, GNS may cache some bindings and skip the Resolver top-hierarchy (this is shown in the bottom part of figure B.13). If, however, the requested resource has been moved, the GNS will not have the correct binding cached, and need to use a Rebind-protocol (together with the requested resource’s last known binding) to get hold of the resource’s new binding.

A sequence diagram (figure B.13) will exemplify how a client’s request for some resource will travel through the architecture:

Figure B.13

The Avaki website can be found at: [http://www.avaki.com/]
B.9 Globus

Participants in virtual organizations often need to share resources, such as storage space, computer cycles, and networks. These resources are usually available only with restrictions based on the requested resource’s nature and the user’s identity. Thus, any sharing mechanism must have the ability to authenticate the user’s identity and determine whether the user is authorized to request the resource. Virtual organizations tend to be fluid, so the authentication mechanisms must be flexible and lightweight, allowing administrators to quickly establish and change resource-sharing arrangements.

The Globus Toolkit, developed within the Globus Project, uses a Grid Security Infrastructure (GSI) to enable secure authentication and communication to meet these requirements. The GSI uses public key cryptography as the basis for its functionality. It also offers secure single sign-on and preserves site control over access policies and local security. It also includes a programming interface for the creation of new secure applications.

B.9.1 Architecture

The architecture proposed by Global advocates the use of LDAP to define a standard resource information protocol and associated information model. The Globus Toolkit’s Meta Directory Service defines and implements a secure, scalable architecture for discovering and monitoring resources in a distributed environment. Information is structured in terms of a standard data model, taken from LDAP. Protocols, services, and APIs are layered by function. APIs and toolkits are superimposed on the grid architecture (figure B.14) [GL1]:

- **The Fabric layer.** It provides the resources to which shared access is mediated by Grid protocols.
- **The Connectivity layer.** It defines core communication and authentication protocols required for Grid-specific network transactions. Authentication protocols build on communication services to provide cryptographically secure mechanisms for verifying the identity of users and resources, and should have the following characteristics:
  - *Single sign on.* Users must “log on” (authenticate) just once and then have access to multiple Grid resources.
  - *Delegation.* Users must be able to give their rights to an application, so it is able to access the resources on which the user is authorized.
- Integration with various local security solutions. Each resource provider may employ any of a variety of local security solutions, including Kerberos and Unix security.

- User-based trust relationships. In order to use resources from multiple providers, the security system must not require each of the resource providers to interact with each other.

- The Resource layer. It builds on the Connectivity layer communication and authentication protocols to define protocols for the secure negotiation, initiation, monitoring, control, accounting, and payment of sharing operations on individual resources.

- The Collective layer. It contains protocols and services that are not associated with any specific resource.

- The Application layer. It comprises the user applications that operate within the environment.

Figure B.14

An application developer’s view of the Globus architecture.

B.9.2 Security

The Grid Security Infrastructure software is a set of libraries and tools that allow users and applications to securely access resources. GSI focuses primarily on authentication and message protection, defining single sign-on algorithms and protocols, cross-domain authentication protocols, and
delegation mechanisms for creating temporary credentials for users and for processes executing on a user’s behalf. It is based on a Public Key Infrastructure and uses authentication credentials composed of X.509 certificates and private keys. It builds on and extends the TLS protocols to address most of the issues listed above, including single sign-on, delegation, integration with various local security solutions, including Kerberos, and user-based trust relationships. In brief, a GSI user generates a public-private key pair and obtains an X.509 certificate from a trusted CA.

A central concept in GSI authentication is the certificate. Every user and service on the Grid is identified via a certificate, which contains information vital to identifying and authenticating the user or service.

A GSI certificate (figure B.15) includes four primary pieces of information.

1. **Subject name.** It identifies the person or object that the certificate represents.
2. **Public key.** The public key that belongs to the subject.
3. **CA Identity.** The identity of a Certificate Authority that has signed the certificate to certify that the public key and the identity belong to the subject.
4. **Digital signature.** The digital signature of the named CA.

### Figure B.15

<table>
<thead>
<tr>
<th>Subject Name</th>
<th>Public Key</th>
<th>CA Identity</th>
<th>Digital signature</th>
</tr>
</thead>
</table>

Globus Certificate

If two parties have certificates and both parties trust the CAs that signed each other's certificates, then they can prove to each other that they are who they claim to be (mutual authentication). The GSI uses SSL/TLS for its mutual authentication protocol.

By default, the GSI does not establish encrypted communication between parties. Once mutual authentication is performed, the GSI gets out of the way, to let communication occur without the overhead of constant encryption and decryption.

The GSI can easily be used to establish a shared private key for encryption if confidential communication is desired. A related security feature is data integrity, described in chapter 3 on page 24.
The core GSI software provided by the Globus Toolkit expects the user's private key to be stored in a file in the local computer's storage. To prevent other users of the computer from stealing the private key, the file is encrypted via a passphrase. To be able to use the GSI, the user must enter the passphrase required to decrypt the file containing the private key. The use of cryptographic smartcards has also been prototyped in conjunction with the GSI. This allows users to store their private key on a smartcard rather than in a file system, making it still more difficult for others to gain access to the key.

GSI provides a delegation capability, which is an extension of the standard SSL protocol. It reduces the number of times a user must enter the passphrase. If a Grid computation requires that several Grid resources be used (each requiring mutual authentication), or if there is a need to have agents (local or remote) requesting services on behalf of a user, the need to re-enter the user's passphrase can be avoided by creating a proxy. The proxy consists of a new certificate, with a new public key and a new private key. The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner’s private key instead of a CA. The certificate also includes a time notation (TTL). Proxies also have limited lifetimes, which tell users how long they should be able to accept communication with the proxy. The proxy's private key must be kept secure, but because the proxy isn't valid for very long, it doesn't have to be kept quite as secure as the owner's private key. It is thus possible to store the proxy's private key in a local storage system without being encrypted, as long as the permissions on the file prevent anyone else from looking at it easily. Once a proxy is created and stored, the user can use the proxy certificate and private key for mutual authentication without entering a password [L1].

When proxies are used, the mutual authentication process differs slightly. The remote party receives not only the proxy's certificate, but also the owner's certificate. During mutual authentication, the owner's public key is used to validate the signature on the proxy certificate. The CA's public key is then used to validate the signature on the owner's certificate. This establishes a chain of trust from the CA to the proxy through the owner.

The Globus website can be found at:
http://www.globus.org/
Appendix C: P2P development platforms

There are three main competing development platforms, JXTA from Sun Microsystems, .NET from Microsoft, and PtPTL from Intel. Intel and the JXTA founders have as their vision to create an open set of standards that will allow interoperability among other peer-to-peer applications and tools. This set of standards is later planned to become a foundation for the next (3:rd) generation Internet. It is most likely that the future standard for the 3:rd generation Internet will be some combination of the techniques from these frameworks, together with regular client-server technologies.

The evolution of these platforms will continue, while their constructors continuously are adding new functions and security features to the environments.

There are some similarities between the platforms. Both JXTA and .NET use XML to connect their internal services. But the use of XML alone does not make them quite comparable. They have completely different fundamental purposes. .NET is focusing more on the traditional client-server architecture of service delivery, and JXTA is specialized for P2P networking.

The .NET technology can, nevertheless, form a rich foundation for P2P applications. Creating a full P2P solution with .NET will, however, require some initial extra work. The developer must specify all of the core P2P interactions. This involves the creation of some of the mechanisms already defined in the JXTA protocols. For example, peer discovery mechanisms, which are not included in the .NET Framework at this time.

C.1 JXTA

Project JXTA began in the summer of 2000 as a Sun Microsystems P2P research project. Although Sun was behind the introduction, JXTA is now an open-source initiative and not a proprietary effort by a single vendor. The open-source development method has been proven to be very efficient in the aspect of creating secure applications. Bugs are rapidly located and corrected by some of all the participating developers. Open-source also motivates the development of lasting security components and do not encourage security through obscurity. Developers will also have the flexibility in removing unnecessary components, or integrating entirely new components like support for smart cards or biometric devices. Often
Authentication in peer-to-peer systems

will the result of an open-source product be more stable and lightweight than commercially developed products. At this time (February 2002) the Project JXTA involves over 8000 members [J1].

C.1.1 Description

Project JXTA was launched in an effort to develop a secure peer-to-peer platform, only using open standards and protocols such as Java and XML. This technology allows a community of peers, or peer groups, to communicate and access data directly from one another rather than depending on some central device such as a server to deliver the information. JXTA creates a very scaleable environment for all participating users, and offers the technology to run applications on cell phones as well as on a desktop computer.

The peer discovery mechanism within JXTA can be completely decentralized, completely centralized, or a hybrid of the two. This is important to be able to support all kind of networks. To be able to locate other peers in the network, JXTA provide support for special peer discovery mechanisms. The current version (1.0) of the framework support the following peer discovery mechanisms [J2]:

- LAN-based discovery, via a local broadcast over the network subset.
- Discovery through invitation. If a peer receives an invitation, the peer information contained in the invitation can be used to discover another peer.
- Cascaded discovery. If a peer discovers a second peer, the first peer can, with the permission of the second peer, view the horizon of the second peer and discover new peers, groups, and services.
- Discovery via rendezvous points. A rendezvous point is a special peer that keeps information about all the peers it knows about. A peer with the possibility to communicate via a rendezvous peer can, this way, find out the existence of other peers. Rendezvous points are good to let isolated peers to get knowledge of other peers within the community, by quickly seeding them with lots of information. To provide, and distribute, the knowledge of well-known rendezvous points (peers), it is possible to let web sites to hold some information of their identity and location.
All peers in a JXTA environment will automatically become part of the global group “NetPeerGroup” when they log on to the system. Communication between peers may be done within this global group, or in some other user-defined group.

JXTA also give peers a possibility to become router-peers, which can be used to overcome the problem where peers are located behind firewalls. Router-peers can both route communication and act as a relay service within the community. The relay service peer is used when direct connection not is possible due to some firewall, or to be able to let peers send asynchronous messages. It permits a peer to temporary store the receiving peer’s message queue until the target-peer connects the relay service and can poll the message queue back.

Nodes within a JXTA community are communicating with each other in a typical UNIX-like manner, including the use of pipes. The pipes in JXTA are unidirectional, which means that one sending pipe and one receiving pipe must be used to be able to establish a two-way communication between two peers.

When juxtaposing the Project JXTA with Microsoft’s .NET, it is relatively clear that Sun not is trying to develop something as big as the .NET Framework. Sun seems to be looking for some more elemental distributed computing technology.

C.1.2 Architecture
The JXTA Framework is not an actual piece of software. It is instead a set of communication protocols with included security. This makes it very flexible and adaptive to most network applications to date.

JXTA, like typical P2P systems, consists of three layers (figure C.1):
- The core layer
- The service layer
- The application layer

The core level consists of the following main protocols [L6]:
- The Peer Discovery Protocol
- The Peer Resolver Protocol
- The Peer Membership Protocol
- The Pipe Binding Protocol
- The Endpoint Routing Protocol
- The Peer Information Protocol
The core layer also includes a security layer, which provides secure (encrypted) communication between the participating nodes. The security layer includes authentication mechanisms to be able to verify the identity of the participating peer. It also provides authorization and access control, to control the different peer’s access rights for each resource in the environment.

Figure C.1

![JXTA 3-Layer Architecture Diagram]

All aspects of JXTA build on XML (the Extensible Markup Language) to structure data as advertisements, messages, and protocols. XML is a good choice for representing data within P2P architectures, and here are some examples why [J3]:

- **XML is language neutral.** Any programming language capable of manipulating text strings is capable of parsing and formatting XML data.
- **XML is simple.** XML uses the text markup method to structure data in much the same way that HTML is used to display text documents in web browsers.
- **XML is self-describing.** An XML document consists of data structured using meta-data tags and attributes, which describes the format of the data.
• **XML is extensible.** XML allows authors to define their own set of markup tags to structure data.

• **XML is a standard.** The World Wide Web Consortium is responsible for maintaining the XML standard, with industry and community input, and has been widely adopted in all areas of the computer industry.

Even though JXTA’s use of XML will specify all aspects of the P2P communication for any generic P2P application, it might not be well suited to use JXTA in some specific stand-alone P2P applications. The network overhead of XML messaging may include more trouble than it is worth, especially in a situation where the application developer has no intention of taking advantage of JXTA’s capabilities to incorporate other P2P services into their application.

**C.1.3 Security**

To be able to enable secure communication within groups, all group owners have the possibility to allow any user to join their specific group, or restrict group membership to only those peers that are authenticated. JXTA includes support for this kind of actions by, for example, letting peers take advantage of the support for digital signatures, and the use of secure connections. The security toolkit also includes TLS at group level, to provide encrypted pipes and provide secure communication from end-to-end.

The current version (1.0) of the JXTA Framework includes a large amount of security primitives, used to support the security solutions in services and applications. It provides an authentication framework, modelled after PAM (Pluggable Authentication Module, first defined for the UNIX platform and later adopted by the Java security architecture, JAAS). PAM methods can be used to integrate login services with several different authentication technologies, such as RSA, DCE, Kerberos, S/Key, and smart card based authentication systems.

It also provides support for a basic, password-based, login scheme that can, like other authentication modules, be plugged into the PAM framework.

At group-level JXTA also supports the use of x.509 certificates, described in Appendix A on page 56, to be able to provide secure group authentication.
Other security issues provided by the security toolkit include cryptographic mechanisms, as a simple crypto library supporting hashing functions (e.g. MD5), symmetric encryption algorithms (e.g. RC4), and asymmetric crypto algorithms (e.g. RSA). Another cryptographic mechanism is the transport security mechanism that is modelled after SSL/TLS, but without the possibility to perform a handshake, or a two-way authentication on a single pipe, as a result of the use of unidirectional pipes.

The toolkit also includes an access control mechanism based on peer groups, where a group member automatically is granted access to all data offered by another member for sharing, while non-members are not able to access such data.

All these security methods included in JXTA’s security layer provide a flexible, and robust platform, which can be used by application developers to build new, secure P2P applications upon.

The JXTA website can be found at:  
http://www.jxta.org/

**C.2 .NET**

.NET, developed by the Microsoft Corporation, is a software development framework for web applications. The frameworks main focus is on the traditional client-server model, but it also provides services that can be used as a platform for building secure peer-to-peer applications.

**C.2.1 Description**

The .NET Framework gives developers and administrators control over their applications and resources security at a high level. It provides an easy-to-use toolset to implement powerful authentication, authorization, and cryptographic routines. The framework also provides developers and administrators to make all critical security decisions (such as what kind of resources that applications should be able to access), instead of letting end users perform these demanding tasks.

There are, however, some design options a developer must consider when designing a peer-to-peer application using the .NET Framework. For one, it is very important to understand how the P2P application will be used. This will make it easier to decide what kind of .NET application models the developer may use. Decisions about the peer-to-peer applications
architecture will also have a significant impact on the type of features that the application will be able to offer. The total range of applications developed with .NET can be thought of as a continuum from pure peer-to-peer to pure client-server systems.

C.2.2 Architecture

- The architecture of .NET (figure C.2) includes a large amount of services, which can be used by application developers to generate various kinds of applications.

Figure C.2

<table>
<thead>
<tr>
<th>VB</th>
<th>C++</th>
<th>C#</th>
<th>Perl</th>
<th>Python</th>
<th>· · ·</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services</td>
<td>User Interface</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ASP.NET</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data and XML Classes</td>
<td>(ADO.NET, SQL, XSLT, XPath, XML)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Base Framework Classes</td>
<td>(IO, string, net, security, threading, text, reflection, collections)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Common Language Runtime</td>
<td>(debug, exception, type checking, JIT compilers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSMQ</td>
<td>COM+</td>
<td>IIS</td>
<td>ADO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Message Queuing)</td>
<td>(Transactions, Partitions, Object Pooling)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

.NET Framework architecture

In aspect of creating peer-to-peer applications, there are four main .NET application models available [NE1]:

- **Web Services.** The Web Services technology provides a way to handle registration, discovery, and content lookup for peer-to-peer applications. It provides the possibility to write a class that listens for incoming requests, process them as they arrive, and send back useful information in the form of objects.

- **Windows Forms.** It is the .NET Framework solution for writing rich Windows-based GUI applications.

- **Web Forms.** Web Forms makes it easy to return HTML content to a peer application. This can be useful to improve the peer-to-peer application with general content about the service or advertisements
about using the service. It also allows a peer to get hold of the latest HTML content from the server during the application start-up.

- **Service Process.** It is used within a peer-to-peer environment as a long-lived discovery server. In cases where the discovery mechanism is not using the HTTP protocol, a service process listening for some other protocol can be very useful.

The framework is predicated on interchangeable parts through the use of XML and SOAP, where XML is the glue that will hold the framework services together. Without this way of putting structured data into a form that can be easily and quickly transmitted and interpreted at the other end, .NET would not be able to work.

### C.2.3 Security

The .NET Framework includes a large variety of security features. For managing user identity, role-based security provides a unified model for authorization and authentication of principals based on identity and roles. Additionally, ASP.NET (another part of the .NET Framework) provides customisation and functionality specifically developed for web application security requirements.

The identity object includes information about the user, or entity, being validated. At their most basic level, identity objects contain a name and an authentication type. The name can either be a user's name or the name of a Windows account, while the authentication type can be either a supported logon protocol, such as Kerberos, or a custom value.

The framework provides support for several authentication mechanisms, which can be used both in P2P applications, as well as in client-server applications. It includes the following main authentication methods [L5]:

- **Basic Authentication.** It sends the username and password to a web server in clear text. Microsoft's Internet Information Server (IIS) authenticates the login against the database of users for the specific domain.
- **Basic over SSL Authentication.** Similar to Basic Authentication, except that the username and password are sent over a secure connection, using SSL encryption.
- **Digest Authentication.** Uses a hashing technique, as opposed to SSL encryption, to send client credentials securely to the server.
• **Integrated Windows Authentication.** Used for intranet scenarios only. It uses the Windows login information of the client for identity verification.

• **Client Certificates Authentication.** Requires each of the clients to obtain a certificate that is mapped to a user account.

In real-life, very few system developers would like to use only the Basic Authentication method because it sends the username and password in clear text through a HTTP channel. A better preferred method would be to put SSL underneath Basic Authentication, to secure the transmitted data. In a P2P environment may also the Client Certificate Authentication method be a good choice, letting each peer to have a specific certificate.

There are also some additional authentication mechanisms supported by the .NET Framework:

• The Microsoft IIS server built-in authentication mechanisms. They can be used to provide authenticated identities to all IIS-hosted applications. If there are corresponding Windows accounts, it can also provide automatic account mapping based on the authenticated identity. The supported authentication mechanisms include Basic Authentication, Kerberos, Digest Authentication, and X.509 Certificates (with SSL).

• Passport authentication, based on the participating user’s Windows account, is another centralized, form-based, authentication service provided by the .NET Framework. It provides a single logon and core profile services for member sites. This benefits the user because no further login is necessary to log on to and access new protected resources or sites.

• Different application developers also have the possibility to write their own custom authentication and authorization mechanisms.

The .NET website can be found at:
http://msdn.microsoft.com/

**C.3 Peer-to-Peer Trusted Library**

The Peer-to-Peer Trusted Library (PtPTL) project, developed by Intel, seeks to lower the bar to security for both small and large commercial software vendors as well as open-source developers.

**C.3.1 Description**

PtPTL is a software toolkit providing security components especially suited for P2P applications. These components include digital certificates,
authentication of peers, secure storage, PKI, digital signatures, and symmetric key encryption.
The toolkit provides the establishment of trust between individual P2P clients as well as the organization of secure groups of trusted peers. It also allows participating users to share information and collaborate securely across the Internet. The main goal of the project is to spur open innovations within the P2P security area. The project will allow developers and users to experiment and prototype a variety of approaches and solutions to security issues within a P2P environment. It can be said to serve as the basis for investigations of the security aspects within different P2P environments.

It has been developed as an open-source project, which is, as mentioned before, an ideal method to use for the development of secure software solutions. The open-source development speeds-up the location and correction of bugs and security holes and will ensure a stable end product. PtPTL provides a secure platform that software developers can use to add elements of security to their already existing peer-to-peer applications. One example of this is the sample application included in the PtPTL release, which shows how security can be added transparently to the Gnutella protocol. Naturally, developers also have the option to produce new secure applications from the ground up using PtPTL. These applications will be immediately interoperable with existing standards, have stable security components, and also be portable.

C.3.2 Architecture
Portability has been a major consideration in the design of PtPTL. It currently supports Win32 and Linux, and can also be ported to some other modern operating system without too much effort. One reason why PtPTL is portable is because it is built upon the open-source OpenSSL toolkit, which provides all of the low-level certificate and cryptographic support within the project. PtPTL provides a high-level interface to the OpenSSL toolkit, which makes Open SSL easy to use and provides the flexibility to support a wide variety of peer-to-peer network models. The library can also be built on top of Intel’s Common Data Security Architecture (CDSA) in place of OpenSSL.

The OpenSSL project is a collaborative effort to develop a robust, full-featured, and Open Source toolkit implementing the SSL and TLS protocols, as well as building a full-strength general purpose cryptography
library. OpenSSL is managed by a worldwide community of volunteers, and can be used for [P1]:

- Creation of RSA, DH and DSA key parameters.
- Creation of X.509 certificates, CSRs and CRLs.
- Calculation of Message Digests.
- Encryption and Decryption with Ciphers.
- SSL/TLS Client and Server Tests.
- Handling of S/MIME signed or encrypted mail.

C.3.3 Security
PtPTL includes a wide variety of cryptographic features through object-oriented interfaces. It supports digital certificates and public key encryption, and a various amount of secure-storage formats. PtPTL conforms to existing cryptography standards like the widely used X.509 digital certificate format, described in Appendix A on page 56, and most of the public key cryptography standards, described in chapter 4 on page 31. It can use digital certificates and public key encryption to authenticate and verify the identity of a remote user over the Internet. The project also allows other methods of peer authentication, such as authentication hardware and biometric devices, to be cleanly integrated into the PtPTL architecture.

The library can be used to digitally sign data, allowing a recipient of a message to identify the sender, or to create digital envelopes (private envelopes of data that only an intended recipient can open).

It also includes some support for networking and operating system primitives (like threads and locks) to allow the creation of complete secure applications.

The project also provides support for other cryptographic security issues, as RSA cryptography, password-based cryptography, privacy enhanced mail format, and various standard symmetric encryption algorithms [P2].

The PtPTL website can be found at:
http://sourceforge.net/projects/ptptl
Authentication in peer-to-peer systems
Bibliography

Literature


Links

[A2] Extend and Accelerate with AVAKI 2.0

[A3] Secure Grid Naming Protocol (SGNP)
(Permission © Global Grid Forum, 5 February 2002. All rights reserved.)

[GR1] Groove Networks, Groove Product Backgrounder
Authentication in peer-to-peer systems

[GR2] Groove
http://www.ece.rutgers.edu/~parashar/Classes/01-02/ece579/slides/groove.pdf, Mars 2002

[GR3] Secure Internet Communication
http://conferences.oreillynet.com/presentations/p2pweb2001/Lowell_A_1748.ppt, Mars 2002

[GL1] The Anatomy of the Grid, Enable Scalable Virtual Organizations

[J1] Project JXTA
http://www.jxta.org/project/www/white_papers.html, January 2002

[J2] Project JXTA: A Technology Overview


[M2] Mangosoft, Secure File-Sharing Services over the Internet


[MA1] Peer-to-Peer Architectures and the Magi Open-Source Infrastructure

[N1] Napster’s homepage
http://www.napster.com/, November 2001
[N2] Marshall Brain’s, HowStuffWorks

[NE1] MSDN, .NET P2P

[P1] OpenSSL
http://www.openssl.org/docs/apps/openssl.html, February 2002

[P2] Source Forge: The Peer-to-Peer Trusted Library
http://sourceforge.net/docman/display_doc.php?docid=3851&group_id=19950, February 2002

[S1] SETI@home, Search for Extraterrestrial Intelligence
http://setiathome.ssl.berkeley.edu, February 2002

[W1] Escher Group, WebRiposte
Authentication in peer-to-peer systems
Glossary

ACL  Access Control List

API  Application Programmers Interface – specifies the interface to which an application programmer can write code.

CA  Certificate Authority

COM  Common Object Model

CRL  Certification Revocation List

DCE  Distributed Computer Environment

DES  Data Encryption Standard

DNS  Domain Name Server

Firewall  A firewall represents a connection to the network that restricts outgoing and incoming requests. This allows users to browse Web pages on external servers, for example, but does not allow connections back to the browsing machine. Some ports may also be open for incoming data, for example to receive e-mail.

GUID  Global User Identity

GSI  Grid Security Infrastructure

HTTP  Hypertext Transfer Protocol

HTTPS  SSL-encrypted Hypertext Transfer Protocol

IP  Internet Protocol

IPSec  Internet Protocol Security

KAS  Kerberos Authentication Server
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KDC</td>
<td>Key Distribution Center</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LOID</td>
<td>Location-independent Object Identifier</td>
</tr>
<tr>
<td>MDNS</td>
<td>Magi’s Dynamic Name Service</td>
</tr>
<tr>
<td>NAT</td>
<td>A Network Address Translators (also known as proxy) provide a mechanism for multiple machines to use a single public Internet address. The NAT creates a private network for the local machines and handles the translation of addresses between a machine on the private network and the Internet at large.</td>
</tr>
<tr>
<td>Nonce</td>
<td>An identifier or number that is used only once, to prevent replay messages.</td>
</tr>
<tr>
<td>PAM</td>
<td>Pluggable Authentication Module first defined for the UNIX platform and later adopted by the Java security architecture, JAAS.</td>
</tr>
<tr>
<td>Passphrase</td>
<td>A collection of words used for authentication instead of a single password.</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>RSA</td>
<td>A public-key encryption and authentication algorithm.</td>
</tr>
<tr>
<td>S/Key</td>
<td>A one-time password system.</td>
</tr>
<tr>
<td>SET</td>
<td>Secure Electronic Transaction</td>
</tr>
<tr>
<td>SGNP</td>
<td>Secure GridNaming Protocol</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Security Socket Layer</td>
</tr>
<tr>
<td>TGS</td>
<td>Ticket Granting Server</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Security Layer</td>
</tr>
<tr>
<td>TTL</td>
<td>Time To Live</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WebDAV</td>
<td>Web Distributed Authoring and Versioning provide facilities for remote document manipulating and editing.</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language, which is an industry standard that has its origin in SGML (Standard Generalized Markup Language). XML is simply the specification for defining tags and attributes.</td>
</tr>
</tbody>
</table>
Authentication in peer-to-peer systems
## Index

<table>
<thead>
<tr>
<th>Letter</th>
<th>Terms</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>.NET, 44, 104</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 105</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description, 104</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 106</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>access control, 21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ACL, 21, 37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>asymmetric algorithm, 31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>authentication, 19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>authorization, 21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Avaki, 37, 86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SGNP, 91</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>basic authentication, 33, 37, 44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>biometrics, 35</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CA, 29, 32, 56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>client-server, 5, 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>CRL, 32, 61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>cycle sharing, 9</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>data integrity, 24</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Diffie-Hellman, 27, 70</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>firewall tunneling, 17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Freenet, 12, 68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 70</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>Globus, 39, 94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gnutella, 7, 15, 65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Groove, 24, 35, 76</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 78</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>instant message, 17</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>JXTA, 43, 99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 101</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description, 100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 103</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>KAS, 53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KDC, 27, 30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Kerberos, 31, 37, 39, 44, 45, 53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>description, 53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>key distribution, 27</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>Magi Enterprise, 36, 81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mangomind, 34, 71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>architecture, 71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>security, 72</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MD5, 33, 37, 44</td>
<td></td>
</tr>
</tbody>
</table>
Authentication in peer-to-peer systems

N
Napster, 15, 33, 63
architecture, 63
security, 64

O
OpenSSL, 45

P
P2P standards, 21
peer-to-peer, 5, 6
architecture, 10
definition, 6
hybrid, 14
pure, 15
platform, 43
private key, 27, 30
PtPTL, 45, 107
architecture, 108
description, 107
security, 109
public key, 28, 31, 34, 35, 36, 38, 39, 44, 45, 56

S
sandbox, 23
security, 17
SETI, 9
SGNP, 38
SSL certificate, 37
symmetric algorithm, 29

T
TGS, 54
trust, 18

V
vCards, 35

W
WebRiposte, 34, 73
architecture, 74
security, 75

X
X.509, 56
architecture, 59
description, 56
XML, 43
Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/