Usage of OpenPGP with mobile agents in peer-to-peer networks

Jorge Marx Gómez, Daniel Lübke,
Department of Computer Science
Technical University of Clausthal
38678 Clausthal-Zellerfeld, Germany
Julius-Albert-Str. 4
Tel: +49-5323-67-18386
Fax: +49-5323-67-11216
{daniel.luebke,gomez}@informatik.tu-clausthal.de

Abstract

Much research effort has been put into the development of mobile agent systems based on peer-to-peer network technology. However, not a very strong focus has yet been on the security of these systems. Only rudimentary security measures have been implemented which are not suited for developing open and anonymous networks which are very popular today, like for example Kazaa. This paper proposes the use of the OpenPGP standard for encryption and digital signatures allowing much more flexibility and fine grained user control of the security settings.

Keywords: OpenPGP, Mobile Agent, Encryption, Signature, Peer-to-Peer Network, X.509

1 Introduction to Mobile Agents

There are lots of different definitions what an agent is. Within this paper we assume that an agent is a process that is able to autonomously initiate changes within its environment and react to changes therein [1, pp. 202]. This definition matches a lot of process types but contains the important characteristics which distinguish agent-bases systems from “standard” designed software. These characteristics are summarized in table 1.

Property	**Description**
Autonomous	Can act independently
Reactive	Reacts in time to changes in its environment
Proactive	Initiates actions which are changing the agent’s environment
Communicative	Can exchange information with other systems, including but not limited to users and other agents

Table 1: Characteristics of an agent(compare [1, p. 202])

The Foundation for Intelligent Physical Agents (FIPA) has released lots of specifications dealing with agent-based systems and how they should be designed.

A very important aspect of this is the communication language: All agents who want to be able to communicate which each other need to agree on a language. A standard for such a language is the FIPA Agent Communication Language (ACL) [2] which specifies a standard language for inter-agent communication.

The use of software agents is also propagated as a solution to common software engineering problems [3].

However, really powerful agents become as mobile agents. Mobile agents are a mixture of mobile code and agents which allows them to travel between different systems.

In classical client/server- and multi-tier- applications only data will be sent over the network. However it may be beneficial or necessary to transfer code, e.g. an agent, and execute it on another computer. This process is called relocation or migration. Advantages of using mobile code and agents are [1]:

- Migrate processes to machines which have unused resources available to distribute the load.

- Execute code near the data or the input, which means that less data needs to be transferred over the network thus enhancing response times.

However sending and execute code and mobile agents on other machines raises some concerns:

- The source of the code needs to be authenticated or executed in a secure environment so that no damages to local data and/or other resources may occur.

- The migration possibly shall take place between different platforms which means further efforts have to be made to make the code portable.

- Currently lots of platforms are being developed, which allow the use of mobile agents, like D'Agents [4]. A list can be found under [5]. Furthermore, mobile agents based systems are being deployed for simulation and load balancing [6].

2 Peer-to-Peer Networks

The term peer-to-peer-network (p2p-network) has a wide range of definitions which sounding nearly identical but sometimes have very different emphasis.

What is common to all definitions is the fact that a p2p-network is a network of devices which are able to access resources on all other devices as well as providing resources to them.

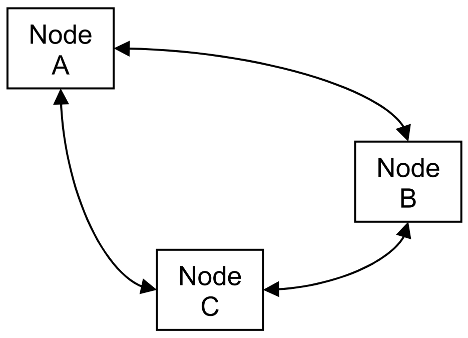


Fig. 1: A small peer-to-peer network where every node can directly access all other nodes

The main issue at where most definitions significantly differ is the question whether this network may have some central services like servers [7] for special supporting purposes, especially name resolution, or not [8].

Examples for p2p-networks include ICQ [9] (instant messaging, chat) and Napster (file sharing) for p2p-networks with centralized lookup-services on the one hand and Windows Networking (file and print services) and Gnutella (file sharing) without centralized resources on the other hand.

The different implementation of how to resolve name shows a real problem when designing a p2p-network: Because these networks are normally designed for applications where nodes can join and leave and may rejoin with different IP addresses and consistently with different DNS names, a p2p-network needs to design and manage its own namespace. This resource centric addressing, like your chat nickname, is something which may be seen as one of the greatest changes and perhaps benefits when using p2p-networks [10].

A common myth is that p2p-networks only scale to a dozen network nodes [11]. This may be true for some kind of software, like Windows Networking, but examples like Gnutella clearly demonstrate that there may be thousands of computers connected within a p2p-network. However scalability within p2p-networks has always been a very critical point and there has not been a golden answer for this problem.

Within the last years, p2p-technology has become generally known through the rise of file-sharing software, although there are many new developments like distributed search engines DFN S2S [12] designed by Germany's National Research and Education Network (Deutsches Forschungsnetz, DFN) and many other prominent ones, like instant messaging, which are also relying on p2p technology.

P2P networks have the advantage, that if one node fails, not the entire network goes down, as well. For example, if in a client-server system, the central server fails, the whole network is not usable any more. In p2p networks, if one node fails, all others are able to proceed and only resources available exclusively from that node are not accessible.

Furthermore, Resources can be distributed, so that failover support can easily implemented, for example, files can be replicated over many nodes, so that when one node is down, the file is accessible through another node. Another advantage is, that resources can be cached or replica can be stored dependent on the network load and performance, so that nodes can query nearby nodes to quicker receive resources.

P2P Solution	**Type**	**Homepage**
ICQ	Chat	www.icq.com
Gnutella	File-Sharing	
eDonkey	File-Sharing	
S2S	Fulltext Search	s2s.neofonie.de
Windows Networking	File- and Printservices	www.microsoft.com

Table 2: Examples of p2p solutions

3 Existing Solutions for Security

For ensuring secure communication between two hosts systems based on mobile agents have to deploy a security concept depending on the applications' needs. Encrypting data ensures privacy, while digital signatures provide authentication and protection against tampering. Because the key-exchange is much easier with public key algorithms, they are widely used.

Public key cryptography uses two keys, one called the public and the other the private key. Both keys are complementary: Data encrypted with one can only be decrypted with the other and vice versa. The public key may be distributed over insecure channels to all entities with whom you want to communicate. The public key is then used by the sending entity to encrypt the data which are then send to the receiver who has the corresponding private key with which he is able to decrypt the data. For signing data digitally the sender takes his private key and encrypts a hash [description hash, footnote] of the data. The encrypted hash is send with the data to the recipient which can decrypt the hash with the sender's public key and compare the hash of the sent data with the received hash.

The public and private keys are generated together. Theoretically it is possible to calculate one key if you have the other but this problem is mathematically so hard that in practice it is impossible.

Probably the most known algorithm for public key encryption is RSA - named after its developers Ronald Rivest, Adi Shamir, and Leonard Adleman.

But a new problem arises: the question whether a key really belongs to the entity it should belong to, for example, because a name is stored with the key. You need to trust a key before using it, so that you do not encrypt traffic to a man in the middle. Therefore a validation process has to be established which leads to a public key infrastructure where users and software can retrieve and validate keys.

Many systems, which are designed for deploying mobile agents [13], utilize the X.509 certificate standard for storing and managing keys in the underlying public key infrastructure, which is defined in RFC 2459 [14].

X.509 depends on a centralized architecture where so called Certification Authorities (CA) sign keys. The key is trusted, if you trust the CA and the CA has issued the certificate containing the key. Examples for CAs are VeriSign [15] or TC TrustCenter [16]. The CA may also sign another entity and allow it to sign other keys as well. This chain of CAs and the key is called a trust path or certification hierarchy. The public key together with user information like name, e-mail etc., and the complete trust path are stored in a so called certificate. The certificate may also contain information about what a user is allowed to do. These may be technical, like acting as a CA, or from a economic view, like ordering goods for a company. However, the X.509 standard is very unprecise here which has lead to differing and incompatible implementations how to handle these extensions. This and many other drawbacks of the standard are discussed in [17].

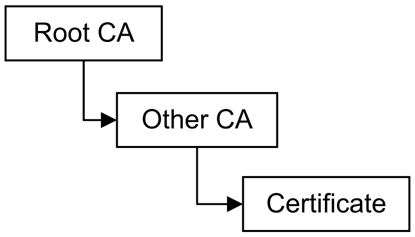


Fig. 2: Principle use of Certification Authorities

The X.509 standard is widely used in industry for storing keys which are used for securing web transactions (Secure Socket Layer (SSL) [18]) or for sending e-mails (S/MIME, RFC 2633 [19]) and are tightly integrated in standard software like browsers and e-mail-software. However, the central approach has some drawbacks:

- if the CA's private key is exposed, the whole security model collapses,
- a key can only be signed by one CA,
- only CAs can sign keys,
- for each CA the certificate has to be installed locally,
- implementations of peer-to-peer networks without any central resource are not possible.

4 The OpenPGP Standard

4.1 Introduction

In 1991 Phil Zimmermann released the first version of Pretty Good Privacy (PGP). It should make the use of encryption easy and usable for private persons as well. It uses strong public key algorithms for encryption and digital signatures. Because of this and the legislation in the United States of America in that time, PGP faced lots of legal trouble, which were all resolved. PGP was further developed and the program changed ownership a lot. Since 2002 it is owned by PGP.com [20] which in turn belongs to Phil Zimmermann [21].

The message format has been slightly changed and standardized as RFC 2440 [22] in the year 1998 as OpenPGP. Today it is widely used in the academic and open source area to encrypt and sign e-mail traffic and software packages.

The OpenPGP standard allows the use of many encryption and signing algorithms and allows the extension by any new algorithm. Commonly supported ones for encryption and digital signatures are:

- RSA,
- DSA,
- ElGamal.

A key should uniquely be identified by its key fingerprint. The fingerprint is a hash value, consisting of 128 bit, which is hard to construct and should be unique world-wide.

However, for simplicity, for normally referencing keys, so called key ids are used, which are the last 4 bytes of the key fingerprint. These are not unique world-wide [23], but can used practically without causing too much headaches.

The keys are normally distributed via so called key servers. These servers contain a database of all known keys and, in general, are syncing their key repositories with each other. One key server can be found under [24]. Users submit their keys to the servers in order to make it possible for others to find and retrieve keys they need.

4.2 Web of Trust

Instead of relying on a hierarchical trust model, OpenPGP uses a decentralized approach which is called the “web of trust”. Its principle is very easy: Everyone can sign any key. By signing one guarantees that the key really belongs to the one, whose name is saved with that key. This practice leads to a graph which represents the trust relationships between the keys and their persons. The vertices represent the persons and keys and the edges represent the signatures and therefore the trust relationships. A real world example is illustrated in figure 1.

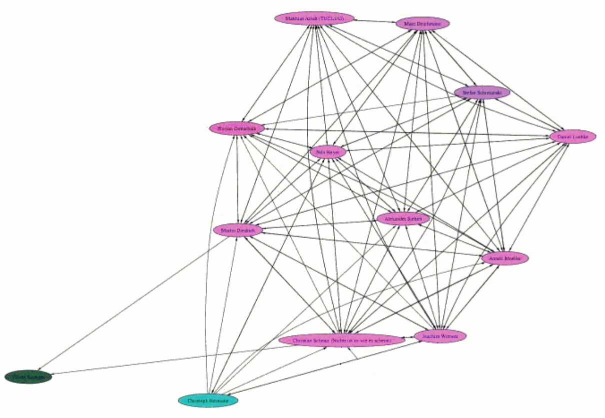


Fig. 3: A small web of trust

To validate a key, the user has to obtain a trustworthy chain between his key and the key he wants to use. This chain is called a path or trust-path. For making this task easy, there are so called pathfinder, like [25].

To establish signatures, keys are signed between colleagues, friends and after so called keysigning parties. A keysigning party is a meeting where people come together in order to exchange their key fingerprints and show their id cards or similar official documents so that everyone can clearly verify that the key really belongs to its assumed owner. Afterwards the keys are getting signed and normally uploaded to the key servers.

The main problem with this approach is, that one often has to trust one or many chains between one's own key and another. The trust in a chain can be controlled via so called “owner trust” which describes to which degree one trusts signatures made by another key: If one thinks, the key's signature are fully trustworthy, that means that a signature really does guarantee that the user carefully verified the key's ownership, one will assign full owner trust to that key. If one is not sure, also marginal or no trust can be assigned.

This way, it is possible to establish CAs in the OpenPGP world as well. The standard proposes an ultimate owner trust, which normally is only assigned to one's own key. This ultimate trust can also be assigned to other keys which in turn means, that this key is as trustworthy as one's own key. So by assigning ultimate owner trust to a key, that key becomes a defacto CA key. There are CAs for OpenPGP keys as well, like HeiseCA [26], which can be treated as a CA but can also be treated as a key like any other, depending on the user's trust settings.

However, it is not possible to inherit trust from a CA: With X.509 a CA may certify another CA. If one trusts the top-level CA, one will also trust certificates issued by the lower-level CA, which automatically is not possible in OpenPGP because the user has full control about the trust settings.

The more keys a verified in that peer-review process, the securer is communication is getting. The web of trust is getting more complete; in it's perfect form, everyone verified everyone and in turn signed every key. There are statistics available for the web of trust, as well as for individual keys. One very prominent example can be found under [27].

4.3 Software and existing solutions

The OpenPGP standard is widely implemented. The most prominent examples are PGP and the GNU Privacy Guard (GPG) [28] which is an open source development, which also was funded by the German government and ported to a variety of platforms, like Windows, Linux, MacOS X, etc. Both implement the cryptographic algorithms and the standard message format. PGP also allows integration into the most famous, commercially used e-mail clients, while GPG uses plugins, developed by 3rd parties for integration.

GPG integration is very famous within open source applications. Native support, for example, is provided by KMail [29], the e-mail client of the K Desktop Environment [30]. Examples for plugins are Enigmail [31] for Mozilla [32]/Netscape Messenger [33] and Thunderbird [34] e-mail clients, as well as the GData plugin [35] for Outlook [36], which is not distributed under an open source license. Enigmail's integration into mozilla is shown in figure 2.

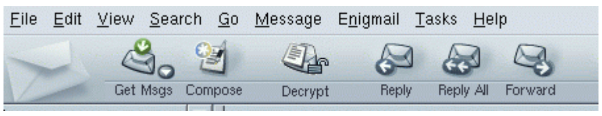
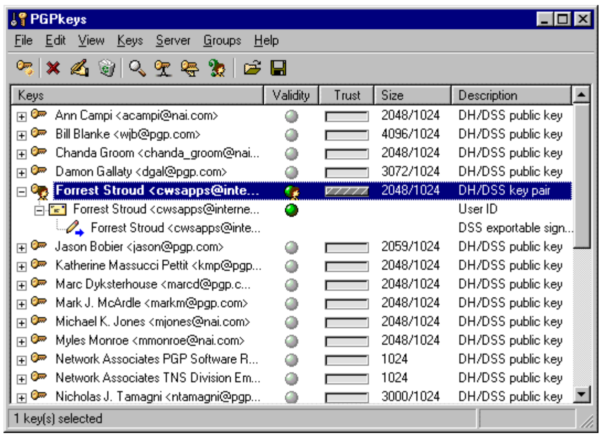


Fig. 4: Enigmail integration into Mozilla (see [31])

PGP has a own graphical user interface for managing keys, which is shown in figure 3.

In contrast, GPG allows the full control of the program from the command-line, however, there are graphical interfaces available, like the GNU Privacy Assistant (GPA)[37] or the GPG Shell [38].



Keys	Validity	Trust	Size	Description
Ann Campi <acampi@nai.com>	●	▭	2048/1024	DH/DSS public key
Bill Blanke <wjb@pgp.com>	●	▭	4096/1024	DH/DSS public key
Chanda Groom <chanda_groom@nai...	●	▭	2048/1024	DH/DSS public key
Damon Gallaty <dgal@pgp.com>	●	▭	3072/1024	DH/DSS public key
Forrest Stroud <cwsapps@inte...	●	▨	2048/1024	DH/DSS key pair
Forrest Stroud <cwsapps@interne...	●			User ID
Forrest Stroud <cwsapps@inte...				DSS exportable sign...
Jason Bobier <jason@pgp.com>	●	▭	2059/1024	DH/DSS public key
Katherine Massucci Pettit <kmp@pgp...	●	▭	2048/1024	DH/DSS public key
Marc Dyksterhouse <marcd@pgp.c...	●	▭	2048/1024	DH/DSS public key
Mark J. McArdle <markm@pgp.com>	●	▭	2048/1024	DH/DSS public key
Michael K. Jones <mjones@nai.com>	●	▭	2048/1024	DH/DSS public key
Myles Monroe <mmonroe@nai.com>	●	▭	2048/1024	DH/DSS public key
Network Associates PGP Software R...	●	▭	1024	DH/DSS public key
Network Associates TNS Division Em...	●	▭	1024	DH/DSS public key
Nicholas J. Tamagni <ntamagni@pgp...	●	▭	3000/1024	DH/DSS public key

1 key(s) selected

Fig. 5: PGP screenshot [see 39]

5 Suggestions for Implementation

The OpenPGP standard is well suited for use in p2p networks. It allows a security model which is not dependent on central resources like CAs. Instead the users can choose and verify which keys and users are trustworthy.

By further bringing the web of trust to the world of mobile agents, new applications are possible: Applications in which agents can learn trust, p2p networks of agents which are operating in an open manner etc.

The question which therefore needs to get resolved is what things need to be encrypted or signed and on what things can security policies work and decide what rights an agent has on the system or within a transaction.

The proposed security architecture is based on three signatures for the agent: One signature for the code, which identifies the programmer of the agent. The next signature identifies the agent's owner. Furthermore, hosts have to sign the agent's state when it is sent over the network.

The rights an agent has within transactions, for example, on line auctions, is stored within an agent passport, in which the owner can state what limits an agent has.

Every participant in the p2p network can have a key: a host, a user and an agent. The keys are correspondingly named host key, user key and agent key.

The host keys and agent keys are identified by a prefix in their description: HOST: and AGENT:, like "HOST: myhost". It should be taken care that these two key types are never uploaded to the central key servers.

Instead their distribution should be part of the p2p network, so that the central key servers are not polluted by keys belonging to virtual entities, which are not part of normal e-mail communication. The user keys can be uploaded to the servers and used for normal e-mail traffic as well.

5.1 Signatures verifying the agent's code

The code of transferred agents can be signed. This is similar to the signing of Java Applets [40] or ActiveX controls [41]. The signature verifies the programmer of the code. Security policies can use this information to check whether trustworthy programmers developed an agent or not.

For instance, a small text file can be send along with the agent, containing file names and their hash values, all signed by the programmers' key, belonging to a human being or a company:

```
--- BEGIN PGP SIGNED MESSAGE ---

SearchAgent.class: 235476AAB7C5D35A
QuickSearch.class: 8306B56D0B2A34CA
--- BEGIN PGP SIGNATURE
...
--- END PGP SIGNATURE ---
```

The programmer's key is only used for verifying the signature. No communication should be necessary between the host system and the agent on the one hand, and the programmer on the other hand.

5.2 Signatures verifying the agent's owner

The owner of an agent is the person or system which sent an agent. The owner is not necessarily the programmer of the agent, because the agent can be developed by a 3rd party and send on demand by a person, for example, for searching the network.

The ownership can be proved by a text file, which contains the agent's name, and the files, belonging to this agent. The files are signed by the programmer as well as the owner, so that the programmer cannot exchange code while the agent is traveling. An example file could look like this:

```
--- BEGIN PGP SIGNED MESSAGE ---

SearchAgent.class: 235476AAB7C5D35A
QuickSearch.class: 8306B56D0B2A34CA
--- BEGIN PGP SIGNATURE
...
--- END PGP SIGNATURE ---
```

Because the signature contains the key id as well, the owner can be identified and results, orders etc. can be send encrypted to him.

It is possible to include the key in this file as well, for example as an ASCII dump, so that the key is automatically distributed with the agent.

5.3 Signatures verifying the agent's data

The internal state of the agent needs to be send from host to host, wherever the agent travels. The host need to sign the data as they pass them to other hosts, so that no data corruption or tampering can occur. The target node can then verify, if the agent comes from a trustworthy node or not.

5.4 Agent passport

In some application scenarios, the agent needs to provide the rights it has, for example, how high the value of acquired goods may be. In X.509 certificates these permissions can be stored and certified, which is directly not possible in OpenPGP. Consequently, this functionality has to be added. The agent passport stores the rights the agent has. It must at least include the agent's unique name in the network and has to be signed by its owner.

The passport can have an start and an expiry date, which can be used to limit the lifetime of the agent's permissions and making it harder to tamper or forge with the passports.

It is possible to store one time passwords or transaction numbers (TANs) in the passport, making it necessary to encrypt it. Using the TANs, e-commerce applications can be implemented easily: The host can decrypt the passport and do transactions on agent's behalf, authenticate itself using the TAN.

By supplying many passports encrypted to different host keys, it is possible to extend this scenario to have different rights when running on different nodes, which may be not equally trustworthy. Furthermore, this allows different set of TANs for different nodes on the network.

An agent passport may look like this:

```
--- BEGIN PGP SIGNED MESSAGE ---

Agent Name: trully.in.tu-clausthal.de
Passport Start: 2003-01-01 00:00
Passport End:   2003-01-02 00:00
TAN: pktncAN4ccs, ujgvhre§$s2, 4c§jfS
MaxAmount: 1000$
--- BEGIN PGP SIGNATURE
...
--- END PGP SIGNATURE ---
```

5.5 Encryption

Preferably the agent, his state and all other information should be transmitted not only signed but encrypted as well. Host-to-host encryption is no problem and many solutions are already implemented, like SSL.

In [42] a solution is proposed specifically for security in mobile agent networks. The idea is quite simple, yet powerful: The agent package or parts of it, are encrypted with many recipients. The agent is then routed from the first to the second recipient and so on. Only hosts on that route can read and execute the agent, because they are the only ones who can decrypt the agent package. Especially with custom passports this solution is very powerful. Since OpenPGP allows encryption to many recipients, this functionality can easily be implemented.

However, the agent's route has to be known before, so this method is not an option for randomly searching a network or for following or not deterministic route.

6 Comparison between OpenPGP and X.509

Although OpenPGP and X.509 deploy the same cryptographic algorithms and are therefore equally technically secure, both are using a completely different trust model.

This leads to some differences: In the OpenPGP world the user has more control over his security settings. There is not necessarily a CA and keys can be verified as well as certified by more than one person. In turn, the user has more responsibilities.

The CA model in X.509 public key infrastructures represents a single point of failure, but allows easy key distribution because on the clients no additional options have to be edited.

The CAs advantage is, that a certificate is either fully trusted or not trusted at all, depending whether the CA is trustworthy or not and the certificate is valid or not. In OpenPGP keys can be marginally trusted because of the paths through which the keys are being validated. There might not always be a short path between two keys making the decision, if a key is trustworthy or not ,very difficult.

An advantage of the OpenPGP model is the fact, that the key validation is not commercialized. CAs are normally getting paid for issuing certificates, but the key-signing process with OpenPGP is free.

While X.509 certificates include a number of optional fields, allowing the extension of the standard to contain information about the rights of an agent, this information has to be transmitted seperately. While this might seem as a disadvantage, it certainly is an advantage: If you want to change the agent’s permissions or want to have different rights depending on the host system it runs on, the use of seperate agent passports is easier.

For both X.509 certificates and OpenPGP keys implementations are available, both commercial or open source, so that they can be easily be integrated into applications and both systems have proved their strength in day to day applications. However, the approach of the web of trust is more suited to the world of p2p networks: No central resources are needed and the security infrastructure is as easily extensible as the network itself. Each node can choose their individual security settings as well as nowadays users can choose which files to share on a file sharing network.

7 Conclusions and Outlook

By using the OpenPGP standard in p2p networks with mobile agents, security can be enforced. Open source utilities like GPG already implement the needed cryptographic functionality and can be used without any licensing costs.

Furthermore the standard allows very flexible trust relationships between all entities participating in the p2p network: Agents can sign each other, users can sign agents and hosts can validate everything as well as signing their network traffic.

Once implemented, new application types are possible: File-sharing networks could be extended in a way to allow mobile agents to move through the network, searching for resources and returning better results. These agents can be anonymous as well as identified through trustworthy signatures without any corporate or central instance controlling the network or enforcing additional costs.

In terms of research, it is possible to develop agents, which are learning which agents and hosts are trustworthy and in turn sign them, so that a flexible trust model is working efficiently. Consequently, learning trust is a field where further research can occur.

References

[1] Verteilte Systeme, 2003, Maarten van Steen, Andrew S. Tanenbaum, pp. 186, pp. 202

[2] FIPA Agent Control Language, Foundation for Intelligent Physical Agents, 2001, http://www.fipa.org/specs/fipa00061/

[3] Agent-Oriented Software Engineering; Nicholas R. Jennings and Michael Wooldridge, 1999

[4] D'Agents: Mobile Agents at Dartmouth College, Center for Mobile Computation, 2002, http://agent.cs.dartmouth.edu

[5] The Mobile Agent List, Fitz Hohl, 2003, http://mole.informatik.uni-stuttgart.de/mal/preview/preview.html

[6] An Agent based approach towards home automation control, Tiago S., 2003

[7] P2P (peer to peer), Network World, Inc., 2002, http://www.nwfusion.com/links/Encyclopedia/P/654.html

[8] On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing, Ian Foster & Adriana Iamnitchi, 2003, http://iptps03.cs.berkeley.edu/final-papers/death_taxes.pdf

[9] ICQ.com - Get ICQ instant messenger, chat, people search and messaging service!, ICQ Inc., 2003, http://www.icq.com

[10] What is P2P... and what isn't, O'Reilly & Associates, 2000, http://www.openp2p.com/pub/a/p2p/2000/11/24/shirky1-whatisp2p.html

[11] peer-to-peer network, CNET Networks, Inc., 2003, http://www.cnet.com/Resources/Info/Glossary/Terms/peer.html

[12] Was ist S2S? - Übersicht , DFN, 2003, http://s2s.neofonie.de/index.jsp

[13] Secure Mobile Agents, Ulrich Pinsdorf, 2003, http://www.inigraphics.net/publications/topics/2003/issue1/1_03a10.pdf

[14] Request for Comments: 2459, The Internet Society, 1999, http://www.ietf.org/rfc/rfc2459.txt

[15] Verisign Inc., 2003, http://www.verisign.com/

[16] TC Trustcenter, 2003, http://www.trustcenter.de

[17] X.509 Style Guide, Peter Gutmann, 2000, http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt

[18] Secure Socket Layer and Transport Layer Security, Liisa Erkomaa, 1998, http://www.tml.hut.fi/Studies/Tik-110.350/1998/Essays/ssl.html

[19] Request for Comments: 263, The Internet Society, 1999, http://www.ietf.org/rfc/rfc2633.txt

[20] PGP Corporation, 2003, http://www.pgp.com/

[21] PGP History, PGP Corporation,2003, http://www.pgp.com/company/pgphistory.html

[22] Request for Comments: 2440, The Internet Society, 1998, http://www.ietf.org/rfc/rfc2440.txt

[23] PGP Keys with Duplicate KeyIDs, Jason Harris , 2002, http://skylane.kjsl.com/~jharris/duplicate_keyids.html

[24] DFN-PCA, DFN, 2003, http://wwwkeys.de.pgp.net/

[25] Experimental PGP key path finder, Jonathan McDowell, 2002, http://the.earth.li/~noodles/pathfind.html

[26] Krypto-Kampagne, Heise Verlag, 2003, http://www.heise.de/security/dienste/pgp/

[27] Keyanalyze report, Jason Harris, 2003, http://keyserver.kjsl.com/~jharris/ka/

[28] The GNU Privacy Guard, Free Software Foundation, 2003, http://www.gnupg.org/

[29] KMail - the KDE mail client, Daniel Naber, 2003, http://kmail.kde.org/

[30] KDE Homepage - Conquer your Desktop!, KDE e.V., 2003, http://www.kde.org/

[31] Enigmail, mozdev.org, 2003, http://enigmail.mozdev.org/

[32] mozilla.org, Mozilla Foundation, 2003, http://www.mozilla.org/

[33] Netscape 7.1, Netscape., 2003, http://channels.netscape.com/ns/browsers/download.jsp

[34] The Mozilla Thunderbird Mail Project, Mozilla Foundation, 2003, http://www.mozilla.org/projects/thunderbird/

[35] GnuPG Plugin, G DATA Software AG, 2003, http://www3.gdata.de/gpg/download.html

[36] Microsoft Office - Outlook Home Page, Microsoft Corp., 2003, http://www.microsoft.com/office/outlook/default.asp

[37] GPA - The Gnu Privacy Assistant, Free Software Foundation, 2003, http://www.gnupg.org/related_software/gpa/

[38] GPGshell, Roger Sondermann, 2003, http://www.jumaros.de/rsoft/gpgshell.html

[39] CWSApps - PGPfreeware Screenshot #2, Jupiter-media Corp., 2003, http://cws.internet.com/screenshots/pgp2.html

[40] Applets, Sun Microsystems, 2003, http://java.sun.com/applets/

[41] COM: Delivering on the Promises of Component Technology, Microsoft Corp., 2003, http://www.microsoft.com/com

[42] Securing Your Data in Agent-Based P2P Systems, Xiaolin Pang et. al., 2003, http://www.computer.org/proceedings/ das-faa/1895/18950055abs.htm