Flexible security in peer-to-peer applications: Enabling new opportunities beyond file sharing

Luciano Paschoal Gaspary a,*, Marinho P. Barcellos b, André Detsch b, Rodolfo S. Antunes b

a Universidade Federal do Rio Grande do Sul (UFRGS), Av. Bento Gonçalves, 9500, Bloco IV, 91.501-970 Porto Alegre, Brazil
b PIPCA/UNISINOS, Postgraduate Program on Applied Computing, Av. Unisinos, 930, 93.022-000 São Leopoldo, Brazil

Received 14 December 2006; received in revised form 7 June 2007; accepted 19 July 2007
Available online 6 August 2007

Abstract

The widespread adoption of P2P applications in environments beyond ordinary file sharing demands the fulfillment of several security requirements. Important steps have been taken towards security in P2P systems, with relevant mechanisms being proposed in the past to address specific vulnerabilities. However, existing approaches lack flexibility, since they do not (include enough mechanisms to) tackle a wide range of requirements in an integrated fashion. In addition, they oblige the user/application to manipulate a complex programming interface, as well as going through a cumbersome configuration process. To address these issues, we present P2PSL (P2P Security Layer), a software architecture that allows gradual and flexible integration of security functionality into P2P applications. To show concept and technical feasibility, we have implemented P2PSL, assessed the overhead it induces, and estimated the feasibility of incorporating the layer into two categories of real world P2P applications.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Peer-to-peer; Security; Middleware

1. Introduction

Peer-to-peer (P2P) applications have gained widespread usage in both academic and corporate environments. Although file sharing and instant messaging applications are the most traditional examples, they are no longer the only ones to profit from a P2P design. For instance, medium-sized applications, whose groups are comprised of tens or hundreds of nodes (e.g., resource sharing [1] and cooperative work [2]), are becoming increasingly common. This is also the case in the corporate arena, where P2P systems allow institutions to exchange services [3].

Although P2P applications can help provide resource sharing and collaboration in a large-scale, wide-area environment with decentralized, loosely
coupled control, their diversification and dissemination are hampered by their current lack of security. It remains difficult to develop P2P applications that address multiple combinations of security aspects, namely confidentiality, authenticity, integrity, authorization, auditing, non-repudiation, reputation, and anonymity. The reasons for that are fourfold. First, existing schemes for securing P2P applications cover only specific (security) aspects (e.g., authentication and reputation) and cannot be easily integrated into a single system. Second, they do not isolate the security aspects from the application. Instead, they oblige the user or the application developer to handle a complex programming interface and go through a cumbersome configuration process.

Third, existing schemes demand a uniform, symmetric behavior of all peers that comprise an application. For some P2P applications, this limitation is undesirable, since the security requirements can vary wildly among its users. To illustrate using a trivial example, consider Skype, the Voice-over-IP (VoIP) application: a given peer may establish different kinds of communication with other peers, each one with its own security requirements (e.g., corporate and home users may have different needs).

Fourth and last, current schemes provide poor or no support for gradual deployment, because they need to be available in all peers of an application. It is very hard, if at all possible, to impose the abrupt adoption of a new security scheme in a large scale, loosely coupled system. Instead, we believe that it is important for the success of P2P systems to allow the coexistence between security-enabled peers and the ones that are not.

This paper presents P2PSL, or Peer-to-Peer Security Layer, which allows the inclusion of security functionality into P2P applications and addresses the lack of integration, isolation, asymmetry, and gradual deployment just mentioned. P2PSL isolates the implementation of security aspects and their configuration from both the P2P application and the underlying communication middleware. Each peer may specify distinct security requirements (complying with different restriction degrees) for each communication channel established with other peers. In addition, the deployment of P2PSL by the peers that comprise the application can be done gradually. The implementation is compatible with JXTA [4], a consolidated set of protocols for P2P systems development, which has been widely used by the community.

The remainder of the paper is organized as follows. Section 2 discusses related work on security for peer-to-peer systems. Section 3 describes typical P2P applications and their security requirements in environments where security in general is desirable. Sections 4 and 5 explain the P2PSL architecture and the peer configuration process, respectively. Section 6 emphasizes implementation aspects, while Section 7 presents performance results obtained with such implementation. Section 8 discusses the feasibility of integrating the P2PSL into real world applications, and Section 9 discusses how P2PSL deals with the most common security threats to peer-to-peer systems. Section 10 closes the paper with concluding remarks and perspectives for future work.

2. Related work

Since the inception of contemporaneous P2P systems, there has been substantial research on related security aspects. As a result, important steps have been taken towards security in P2P systems, with relevant mechanisms being proposed to address specific vulnerabilities. In this section, we cover related work, by first addressing the issue of security in P2P systems in general, and then focusing on the problem at hand.

Despite the breadth of the literature regarding security in P2P, research efforts can be categorized in few classes. One such a class is the analysis of structures and topological properties with respect to security. For instance, in [5] authors state that $k$-regular networks are more robust against attacks because they do not include highly connected “hubs” or nodes with special functionality. This finding is related to another class, availability in P2P systems. It addresses both Denial-of-Service (DoS) attacks [6,7], as well as routing attacks [8,9] like Eclipse [10], whereby a correct node is “eclipsed” from the rest of the network by surrounding, malicious nodes. Such Eclipse attacks rely on an authentication vulnerability inherent with large-scale, decentralized P2P systems: the difficulty to properly authenticate users, preventing a malicious user to get hold of multiple identities (known as the Sybil attack [11,12]). There have been proposals towards secure routing, such as in [13,14], which provide defenses against routing attacks. Authors in [15,16] attempt to overcome additional issues raised by weak authentication without compromising P2P scalability by introducing decentralized Public Key Infrastructures (PKIs). The former paper is based
on quorum, while the latter relies on the Pretty Good Privacy (PGP) web of trust model and the Local Minima Search protocol. Authorization, on its turn, has been addressed in several papers, such as in [17], an approach to control peers joining a P2P group, and [18], an access control architecture for P2P applications based on the Role-Based Access Control (RBAC) model. In both papers, these are isolated implementations that cannot be easily extended or integrated to other security mechanisms.

There has been a lot of research work on P2P to address two other aspects, which are also security-related: anonymity (along with negability) and trust (along with reputation). Regarding the former, FreeNet [19] and Free Haven [20] are two important examples of anonymous P2P systems in current use today. These systems may be based on an anonymous routing infrastructure, such as Tor [21]. Regarding trust and reputation, Marti and Garcia-Molina [22] provide a good survey about trust issues on P2P systems. There are numerous distributed protocols for trust and reputation management; representative examples are EigenTrust [23] and FuzzyTrust [24]. The interested reader may also refer to [25,26], two recent surveys on P2P systems.

We now focus on the specific problem of creating applications that embed a combination of security aspects. The design of new schemes to simplify the development and deployment of secure P2P applications is of paramount importance to expand their use. There have been attempts to achieve that, such as JXTA and PtPTL – Peer-to-Peer Trusted Library [27]. JXTA provides functionality like encryption, signatures, and hashes for the development of secure P2P applications. However, it obliges the programmer to explicitly include and handle security-related code, leading to additional development complexity. PtPTL, on its turn, is an OpenSSL-based API that allows for the establishment of trust between individual peer-to-peer nodes as well as the creation of secure groups of trusted peers. Both JXTA and PtPTL are restricted in relation to the security mechanisms supported: authentication, confidentiality, and integrity. When interested in employing other mechanisms such as non-repudiation, authorization, auditing, and reputation, the application developer has almost no support from the mentioned schemes. The same limitation is observed in most (if not all) of the work on P2P security.

Further, with respect to configuration of security in P2P applications, it is particularly important to adopt more flexible, decentralized, and close-to-user security mechanisms due mainly to scalability concerns (machine user is typically also the administrator). It is not the case of the aforementioned work [17,18], which require policies to be stored and maintained in centralized servers. Approaches proposed by the web service [28] and grid computing [29] communities, in spite of their generality and completeness to secure loosely coupled distributed systems, also fail in this regard. They still depend on a great deal of configuration, which is often performed in a centralized manner.

In an earlier work [30], we introduced P2PSL, aiming at circumventing those limitations. The current paper extends the previous one in many points. It reviews the original design, providing new insights. The authentication scheme is improved, leading to a more scalable protocol. Finally, the paper presents results for a new set of experiments, considering both (elastic) file sharing and (non-elastic) streaming applications.

### 3. P2P applications and security requirements

The amount of related work shown in the previous section demonstrates how important security in P2P has been regarded by the scientific community. This concern is fueled due to (i) the variety of attacks P2P applications are intrinsically vulnerable to [31] (e.g., identity theft by unauthorized or falsely authorized parties, privacy invasion, loss of data integrity, and repudiation of previous transactions), and (ii) the interest of applying this technology in more important, serious activities, such as the ones related to business in the enterprise. In this section, we revisit the main types of P2P applications and discuss security requirements they demand in some use cases.

According to [25,26], peer-to-peer applications can be grouped into five categories, as follows:

- **File sharing:** also named content distribution applications by some authors, file sharing applications allow any user to publish files and disseminate them to a potentially large number of users geographically spread across a very wide area. The content of the published files is read-only.
- **Network storage:** network storage applications provide similar functionality to file sharing applications. However, as opposed to file sharing, in network storage applications the content of
stored files can be changed. If replication is used to improve availability, content alterations must be consistently propagated to all replicas. Furthermore, write operations – and in some cases also read operations – are normally subject to access control and restrictions.

- **Overlay multicast**: overlay multicast applications provide a communication infrastructure to overcome the lack of native network multicast support. They allow a certain content to be efficiently transmitted by a node and delivered to a large number of nodes (users) geographically spread across the globe.

- **Distributed computing**: distributed computing applications aim at intensive processing by consuming idle processing capacity of computing resources that comprise a grid infrastructure (also known as cyber-foraging).

- **Instant messaging and collaborative computing**: these applications allow users to communicate directly through voice (VoIP), text messages, images, etc., without depending on centralized servers.

Several security requirements must be met if the aforementioned applications are used for corporate activities or in any other restrictive scenario. Authenticity and integrity are important to all categories of applications enumerated. Guaranteeing confidentiality is relevant in applications where private information is exchanged with or stored in remote nodes. Sharing confidential corporate information, executing remote corporate backup, and communicating with a customer through a VoIP session are examples of activities, which require effective data protection mechanisms (specially when executed on top of P2P substrates). Authorization is also an important security “building block”, since it allows definition (and enforcement) of a user’s or group’s ability to access specific directories, sets of data, or node resources. This functionality may be demanded not only by the applications already mentioned but also by distributed computing applications, which comprise the use of resources (e.g., CPU, memory, and storage) belonging to remote nodes.

A more relaxed setup or different combination of security mechanisms may be used according to the nature and purpose of the P2P application. For example, the applications listed above could probably be less demanding in terms of security if they were employed for “personal” use. That is what happens with ordinary P2P file sharing applications, which generally adopt looser security mechanisms such as weak authentication and reputation. Other applications such as Freenet [32] and Free Haven [20], on their turn, aim at preserving the anonymity of content provider peers. In this particular context, anonymous routing is used instead of authentication and authorization mechanisms. Coping with different security requirements imposed by P2P applications in a flexible way is, therefore, an important factor in popularizing peer-to-peer theory and applications.

4. **P2P Security Layer (P2PSL)**

In this section, we describe the architecture of the P2PSL. First, we discuss how security modules are combined like puzzle pieces to provide flexible, asymmetric security. Then, we describe the characterization repository, which does the mapping between modules and security requirements. Last, we show how a configuration repository is used to keep track of security requirements specified by both remote peers and the local one.

P2PSL is based on the addition of a security layer which is implemented and configured independently from both the P2P application and the underlying communication middleware. P2PSL acts as a filter, taking outgoing messages from the application, operating on them, and then forwarding them to the lower layer, and conversely for incoming messages. The security requirements are satisfied by modules that implement different security techniques. In line with the intrinsically decentralized nature of P2P applications, the definition and configuration of modules to be employed is done autonomously in each peer by the local user (helped by a graphical interface, as explained later). The configuration of the security layer is based on profiles, each one serving different security needs. So, known peers can be grouped in one of the pre-defined profiles; peers unknown *a priori* are automatically placed in a default profile.

Fig. 1 shows an example of a network of peers using P2PSL. The configurations for Peer 2 and Peer 3 are shown in detail. To illustrate, consider Profile B in Peer 2, which specifies that authentication must be applied to all outgoing messages, and authentication and confidentiality are demanded for all incoming ones. Peer 2 then associates such profile with Peer 3, which means that Peer 2 will employ and require
authentication whenever communicating with Peer 3, as well as confidentiality when receiving from it.

The P2PSL instance associated with the peer during system operation behaves like a wrapper between a P2P application and the underlying communication middleware. Whenever messages are received or sent, the selected modules are triggered to guarantee the security requirements established. Fig. 2 illustrates this process, where Peer 2 employs Profile B to send and receive to/from Peer 3. In this example, outgoing messages are passed to a digital signature module. A module may perform changes in the message according to the security requirement being enforced; this is the case in the example, where a signature is added. Incoming messages, on their turn, are passed to the modules digital signature and cryptography, also according to Profile B. If the message goes through successfully, it is delivered to the application. Otherwise, depending on the characteristics of the module, the message is simply discarded.

4.1. Security modules

The security modules are the key pieces of P2PSL. Each security technique implemented is represented through one of these modules. For example, the message auditing support is typically implemented through a log generation module; authentication and message integrity, through a digital signature.
module; and access control to resources (authorization), through verification of access policies.

The modules are based on the utilization of a generic interface which allows the addition of new modules in a simple manner. Each module has methods for the verification of incoming messages and for the preparation of outgoing messages. When invoked by the security layer, the module does the processing needed (possibly changing message contents) and returns the status informing whether the operation was successful or not.

4.2. Characterization repository

Conceptually, the security modules differ among each other in regards to input parameters and utilization dynamics. An independent repository is used to handle such heterogeneity and avoid integrating directly into P2PSL the characteristics of each idealized module. The repository is implemented through an XML file, which contains the characterization of available modules. It defines the parameters and the usage of each module, as well as the combination of modules that fulfill each of the security requirements.

The parameters of each module must be configured so that it behaves properly. Examples of parameters are the identification of a key to be employed in ciphering or in a digital signature, the level of detail to be used in a log generation module, and the path name of a file with access policies for an authorization module. In addition, some of the parameters specified locally can be important for other peers in the network. For example, if a peer wishes to encode a message using asymmetric cryptography, it needs to know in advance the key employed by the destination peer. Such information is negotiated through a configuration protocol (explained in Section 5). P2PSL associates four basic characteristics (specified through attributes) for each module. The first attribute (export_requirement) specifies if message changes must be performed at the sender or not in order to allow the module to be used at the receiver. For example, in modules that involve techniques like authentication and cryptography, the outgoing message needs to be changed in order to allow the recipient to apply the technique when the message is received. On the other hand, with modules such as the log generation and the verification of access policies, the original content of the message is enough to apply the technique upon receipt. The second attribute (obligatory_if_applied) indicates whether or not it is mandatory to apply the module when the message is received to recover the original data. This is the case with cryptography modules, but not with authentication, as the latter only adds a signature to the message. The third and fourth attributes (allow_on_bcast_sending and discard_on_failure) regard, respectively, the possibility of employing a module in broadcast transmissions (as long as the underlying P2P substrate provides it), and the need to discard a message when it fails the verification process upon receipt.

The repository stores, besides parameters and attributes of each module, the mapping between requirements and security modules. Note that this mapping is not 1:1, since a module can serve more than one security requirement, and a requirement can demand multiple modules. For instance, the combination of modules for verification via MD5 hash and another for signature of this verifier represents an option of message authentication. The following items map the most relevant security requirements and examples of techniques that can be employed in P2PSL:

- **Confidentiality**: PGP cryptography, RSA cryptography, RC4 cryptography, AES cryptography.
- **Authenticity**: PGP signature, RSA signature, RC4 cryptography, Message authentication code, SHA1 hash + PGP signature on digest, SHA1 hash + RSA signature on digest, SHA1 hash + RC4 signature on digest.
- **Integrity**: PGP signature, RSA signature, RC4 cryptography, Message authentication code, SHA1 hash + PGP signature on digest, SHA1 hash + RSA signature on digest, SHA1 hash + RC4 signature on digest.
- **Authorization**: RBAC-based access control.
- **Non-repudiation**: On-line service for evidence generation/verification, PGP signature + time-stamping, RSA signature + time-stamping.
- **Reputation**: Debit-credit reputation, feedback-based reputation, credit-only reputation.

Fig. 3 shows as an example parts of a characterization file (in XML). Lines 2–23 indicate the security modules available. The definitions regarding *PgpEncryption* are shown in detail. Lines 4–7 define the usage characteristics of a module, setting each one of the four attributes previously explained.
Lines 8–17 specify parameters. Lastly, lines 24–31 illustrate the mapping between the set of security requirements satisfied and the existing modules; there, it defines that confidentiality is obtained through the `PgpEncryption` module.

### 4.3. Configuration repository

The configuration repository contains all the information required by the security layer to properly employ the modules specified by the user. The repository is implemented through an XML file and its main role is to store the configuration regarding each profile. Notice that the security requirements and modules to be applied are specified independently for incoming and outgoing communication channels. Fig. 4 provides an example of profile representation; it refers to Profile B of Peer 2 in the scenario shown in Fig. 1. Besides the profile definition, the XML file contains the list of remote peers and their requirements, as well as standard settings for the modules locally available.

Fig. 3. Example of module characterization.

An important attribute set for each profile is named `respect_remote_requirements` (as indicated by line 2 in Fig. 4). When this attribute is enabled, requirements of remote peers (i.e., security modules) are automatically satisfied (using corresponding modules) when messages are exchanged with peers belonging to the corresponding profile. Note that the set of modules applied will be the union of modules locally specified by the profile and the ones demanded by (a profile in) the remote peer.

### 5. Peer configuration

P2P networks are expected to be dynamic, heterogeneous, and asymmetric in terms of security. Because of these properties, it is unfeasible to manually configure the security layer of a peer in regards to every other peer. P2PSL tackles this problem in two ways. First, it lets users to classify peers according to profiles (as presented in the previous section). Second, the security layer includes mechanisms that
guide and automate the configuration process, allowing a large number of peer relationships to be managed effortlessly.

In this section, we build on the description of P2PSL and present the three different configuration moments through which peers may experience: (i) an initial (user) setup that precedes the activation of a peer, (ii) a negotiation whenever the peer enters the network, and (iii) configuration adjustments that occur in response to P2P network changes. Each one is detailed below.

The first moment refers to the initial setup. Before a peer with P2PSL is run, a digital certificate needs to be obtained from a Certification Authority. The CA creates a new certificate with the information provided by the peer and signs it with its own private key before returning to the peer. Thereafter, the peer may provide other peers with its (CA signed) digital certificate. As long as the receiving peer trusts the CA, it can directly accept this certificate, without the necessary involvement of the Certification Authority. This avoids overwhelming the CA with key requests, increasing system scalability. Security-wise, the use of a CA is typically better than a public key distribution authority. A CA is not susceptible to attacks aiming at modifying public keys, because it does not keep a copy of them. Instead, the certificate will be transmitted directly by the peer that wishes to identify itself. Although the scheme presents a good degree of scalability and security, it still presents the potential for fraud. The attack may occur when a peer needs to obtain for the first time or to renew its public key with the CA. When somehow the private key is broken, the user needs to detect such a problem and communicate the CA; however, it may take some time before peers learn that a certificate has been compromised.

When a peer ingresses a P2P network, there is an external bootstrap process in which it finds about other active peers. As far as P2PSL is concerned, a peer needs to determine the set of security mechanisms demanded by each other peer it wishes to communicate with. This corresponds to the second configuration moment. The protocol that drives this communication relies on a pair of communication primitives: send (unicast) and receive, provided by the underlying middleware. The protocol is described below through an example, which is illustrated in Fig. 5. We assume, for now, there are no unknown

```
<profile name="Profile8">
  <respect_remote_requirements="true">
    <incoming_requirements>
      <requirement name="Authentication"/>
      <requirement name="Confidentiality"/>
    </incoming_requirements>
    <outgoing_requirements>
      <requirement name="Authentication"/>
    </outgoing_requirements>
    <incoming_modules>
      <module name="PgpSignature"/>
      <module name="PgpEncryption"/>
    </incoming_modules>
    <outgoing_modules>
      <module name="PgpSignature"/>
    </outgoing_modules>
    <profile_peer_members>
      <profile_peer_member name="Peer3"/>
    </profile_peer_members>
  </profile>
```

Fig. 4. Example of profile configuration.

![Fig. 5. Time diagram representing discovery of requirements of other peers.](image)
failures in the P2P network, and discuss later assumptions and implications about faults. Let Peer 1 be the peer that enters the network, and Peer 2 and Peer 3 peers that are already active.

1. When entering the network, Peer 1 sends a signed Requirements request message to the set of peers whose IP was obtained during the bootstrap. This message is signed with the certificate’s private key, and contains the digital certificate issued by the trusted CA. No matter Peer 2 and Peer 3 had or not previous communication with Peer 1, they can verify the validity of the message through the received digital certificate.

2. Being the request message valid, the peers reply with Requirements request and reply specifying its requirements towards Peer 1 and at the same time asking Peer 1 about its requirements towards each of them. These messages are also signed, and contain the sender’s digital certificate.

3. Peer 1 verifies the signature of Requirements request and reply received from Peer 2. Assuming the message is correct, Peer 1 sends an individual Requirements reply to Peer 2 containing its own requirements towards Peer 2. Peer 1 does the same for Peer 3.

Some of the peers obtained from the bootstrap process and that have responded to the initial transmission may be previously unseen. As already mentioned, the authenticity of messages is ensured by means of digital signatures. If a signed message received does not match the expected digital signature, the peer will ignore it.

Peers that were unknown previously are at first deemed untrusted and automatically placed in one of the two predefined profiles, as follows. The legacy profile refers to peers that do not have the security layer implemented or properly configured (e.g., with an invalid digital signature). The default profile refers to peers that implement the security layer and possess a valid digital signature, but were previously unseen by this peer (e.g., first time the peer encounters this peer id). In Fig. 5, this could be the case if Peer 1 did not know about Peer 3, in which case Peer 3 would be associated with the default profile.

So, at the end of the second configuration moment, a peer will have determined the set of requirements (modules to be applied locally) when sending or receiving to/from every other peer it wishes to communicate with, and will have told these peers which are its own requirements.

The third and last configuration moment refers to the changes in the security configuration of a peer, which can happen at any time during its life. This is more likely in long-lived applications, where there is enough time for new peers to come in or existing peers to change their own requirements. If so, in reciprocity, a peer may wish to update its own requirements towards another peer. This would be typically achieved by moving a peer from one profile to the other. It is also possible to change the requirements associated with a profile, the one, which the remote peer belongs to, but with consequences to the other peers as well.

The introduction of new requirements affects other peers and needs to be communicated to them. When a peer, say, Peer 1, changes the set of requirements towards another peer, say Peer 2, Peer 1 sends a message to Peer 2 informing the new requirements. If the set of requirements regarding Peer 2 is augmented, messages sent by Peer 1 to Peer 2 would be affected immediately. To allow a graceful increase of requirements, a transition interval can be specified by the user, temporarily delaying the application of restricting measures.

In the description above, we assumed there would be no failures in the network or in the peers. Now we consider some of the most common types of failures that can happen in a P2P system, and how P2PSL is affected by them. Naturally, the failure semantics of an application that employs P2PSL is affected by the choice of underlying P2P communication substrate. First, a peer may fail to receive a response for a contacting message from an active peer (whose identity is taken from the bootstrap) due to a network-related failure. To handle this problem, a peer needs to employ timers to prevent indefinite waiting. A peer that fails to respond will not have its requirements registered and, therefore, will remain in the default or legacy profiles (considered unsafe for communication). So, the waiting timer needs to be long enough, since in case of peer or network contention, the arrival of responses can be arbitrarily delayed. Further, a peer may respond to a message and then crash, in which case one peer may have to limit waiting for a message from another peer. Finally, P2PSL has no support for Byzantine failures, when a (potentially trusted) peer starts behaving arbitrarily during operation, maliciously or not.
6. Implementation

We have implemented a fully functional prototype of P2PSL. It is based on the introduction of a SecurePeer wrapper class, instantiated between the P2P application and the underlying communication middleware, as shown in Fig. 6.

The implementation assumes that a message-oriented paradigm is used on the communication between peers. Each send or receive operation initiated by the application is caught by the SecurePeer instance, which passes the message through the configured modules. The modules are specializations of class SecurityModule, which offers a generic access interface that is employed by class SecurePeer. The methods in SecurityModule represent the verification of each incoming and outgoing message (verifyIncomingMessage and adjustOutgoingMessage, respectively) to see if they satisfy the requirements specified. The access to the modules is done solely through this generic interface. Along with dynamic class loading in module instantiation, it makes the security layer extendible: new modules can be added without having to recompile the rest of the layer.

The implementation code, written in Java, is largely independent from the communication middleware. For the experiments described in this article, JXTA [4,33] was used. JXTA is a project that aims to establish a set of implementation-independent protocols that allow the creation of a general-purpose P2P structure, which can be employed by different applications. More specifically, the communication code is based on the interfaces provided by the JXTA Abstraction Layer (JAL – [34]). JAL is a library that abstracts several aspects of the JXTA architecture, offering the programmer a simpler interface to access common functionality in P2P systems, like message transmission (unicast or broadcast), creation of groups or resource search. JXTA is widely used in a variety of projects and research work, allowing our implementation to be useful for several existing systems.

In practice, P2PSL is used as a library. The implementation is provided as an independent Java A Rchive (JAR) file, which allows P2P applications to use its classes by adding the file in the CLASS-PATH environment variable and importing the appropriate (p2psl) package.

6.1. Available modules

Five modules were implemented to provide the following security functionality: authentication, confidentiality, integrity, authorization, and possibility to audit message exchange. New modules can be added with no change or source-code recompilation, such as when incorporating a new security mechanism or choosing an alternative technique more suitable for a given scenario. In line with P2PSL philosophy, the modules were implemented such that peers that do not have the security layer can still participate on the system. This allows a gradual adoption of the security layer in operational P2P systems, and gives the user of each peer the choice of employing or not the layer. Incidentally, peers that implement P2PSL can block messages sent by peers that do not have it by requiring, for example, message authentication from those peers. Next, we present the modules that were implemented.

**PGP signature.** The PGP signature module aims to ensure authenticity and integrity of message exchange between peers. Because the decentralized
PGP exchange key scheme is used, a central certification authority is not obligatory. In this model, public keys are stored in arbitrary servers, or even exchanged directly between the interested parties. The validity of keys is ensured through a web of trust formed by PGP key users: each user signs the public keys known to be valid, to share this information with other users that trust his/her signature.

The implemented module checks existing public and private keys locally available in the node where the peer runs. These keys can be managed through applications already consolidated like GnuPG and Kgpg. The private key used is established previously by the user (through an input parameter), and the generated signature is added to each message sent using a specific field. When messages are received, the signature is verified through the corresponding public key. If the signature is valid, the message can be forwarded to other modules or delivered to the application. Otherwise, and if the peer was configured to require authentication, the message is discarded. The generation of signatures is based on the BouncyCastle library, which implements the PGP algorithms. Note that PGP allows the use of different signature generation algorithms, like RSA and DSA, as well as the specification of the key size. Both characteristics are specified when the pair of keys is created.

It is important to point out that PGP may not be the best choice for general P2P systems. This is because the PGP web of trust model relies on a chained, ad hoc peer certification mechanism, which cannot help secure communication among peers who have no prior social relations (frequently the case in large-scale P2P applications). In addition, the trust chain can be compromised by the treachery of one (or a very few) trusted peers. Despite its weaknesses, the web of trust approach is widely adopted and its replacement with a radically different approach may pose operational and public acceptance challenges. Hence, the natural evolution of our work may encompass the investigation and development of new, hybrid modules – possibly based on [15,16] – to solve some of the drawbacks of PGP-based security mechanisms.

**PGP cryptography.** Like in the previous module, this one employs the facilities provided by PGP to guarantee message confidentiality. The PGP cryptography module available extracts from the outgoing message all fields inserted by the application, and then generates a byte array which is cryptographed and inserted in the message in a specific field. When it is received, this array is deciphered, and the fields of the original message are restructured and reinserted in the message, so that it remains transparent to the application. Like the signature module, routines provided by the BouncyCastle library are used.

**Symmetric encryption.** This module aims to provide a fast mechanism for message confidentiality. Instead of relying on the use of asymmetric encryption algorithms for each message exchanged (like the PGP cryptography module), the same shared, symmetric key can be employed by communicating parties to cipher a set of consecutive messages. This improves efficiency specially on applications that exchange a large number of small messages in a short period of time. The key expires if no message is sent for some time. When so, the peers will need to agree on a new key.

The present implementation uses Java native cryptography library. While it relies upon the AES algorithm with fixed-size keys of 128 bit, P2PSL allows the module to be further enhanced so that the peers negotiating the shared key can also agree on the key size and encryption algorithm.

**Verification of access policies.** Aiming to provide access control to resources (authorization), the module for policy verification uses generic information in the message (like date, message size and sender identification) to define if the message can be delivered or not. Specification and verification of policies are based on XACML, a standard created by OASIS for the definition of access control policies through XML. The policies are defined classifying the peers in roles, following the Role-Based Access Control (RBAC) model. Hence, two distinct repositories are established (both implemented through XML files): one for the specification of access policies, and other to fit the peers in the defined roles. Whenever a message is received, the roles played by the sender are determined and the current policies consulted, based on the information relevant to access control. This query is processed using the Sun XACML library, and returns as result whether the access can be granted. If access is not allowed, the message is silently discarded. The interested reader may refer to [35] for more details on the policy-based access control module.

**Log generation.** When used, the log generation module creates, according to a preset level, a trace that presents information about the exchanged messages. Examples of information are the instant of
each event, characteristics of messages and information about peers taking part in the communication. The output is directed to a file established during configuration. Through this module it is possible to audit the message exchange, identifying problems in the way the application is functioning or being used.

6.2. Configuration assistant

To ease the task of adjusting the configuration repository, described in Section 4.3, a Graphical User Interface (GUI) was created. This GUI or front end is activated during the peer configuration process. For each profile, the user specifies the aspects to be met, and then determine the combination of available techniques to reach the desired goal.

Fig. 7 presents a snapshot of the tool with the security profile configuration screen, where parameters can be specified for a module (in the case shown, a PGP signature). Once completed, the established configuration can be written to the XML file stored in the configuration repository and interpreted by the security layer.

7. Experimental evaluation

The need for security, particularly in corporate applications, is clear. However, the required security mechanisms to be incorporated into a P2P application will introduce overheads. This section presents an experimental evaluation with the implementation of P2PSL. Our aim was to use this implementation as a proof-of-concept of P2PSL as well as make measurements of the latency and network usage overheads induced by the security layer.

Experiments were conducted using a synthetic load in order to isolate and measure the overheads of P2PSL without modules and of each module individually. They were performed in a 2.4 GHz Intel Pentium 4 CPU with 1 GB RAM running Linux. Although we investigated different choices for algorithms and key security parameters, we report only the main results here. To obtain statistically sound results, each experiment was repeated 200 times. Adopting degree of confidence of 99%, the confidence interval seen for any experiment was no larger than 0.38 ms.

Message size is expected to play an important effect into the performance of the security layer, and so the experiments were conducted using message size as a factor: 128, 256, 512, 1024, 2048, 4096, 8192, 16,384, 32,768, and 65,536 B of data, with randomly generated contents. Fig. 8 shows the average latency results for transmission (a) and reception (b) of messages. The values shown in the plots refer to P2PSL configured with no security modules (Empty Layer), and to each module isolated\(^1\) (PgpSignature, PgpEncryption, SymmetricEncryption, Logging, and PoliciesChecking).

Examining the figure, first notice that the empty layer takes around 14 ms in both send and receive operations, with almost no variation regarding message size. As expected, the PGP encryption module

\(^1\) Not including the empty layer overhead.
induces a very high overhead per message, whose average reaches almost 130 ms when ciphering messages of 64 KiB. The overhead will depend on the key size; in the experiments shown, we chose to use 1024 bit. The PGP signature module also induces substantial overhead, reaching 22 ms for the generation and 40 ms for the verification of signatures in messages of 64 KiB. As opposed to the PGP encryption and signature modules, the SymmetricEncryption module – configured to employ a 128-bit encryption key – performs much better, generating an overhead of around 1 ms for messages smaller than 8 KiB. The module for access policy verification (receiver only) presents low overhead, around 2 ms for a single rule, but the delays will be larger according to the number of policies to be verified. The module for log generation induces a small overhead, around 4 ms, regardless of message size, because in the experiments the level of detail was set to “medium” and thus message contents were not written to the log.

Overall, individual delays were large, typically within tens or hundreds of milliseconds per message. These delays should not be considered alone, but instead combined, since a given P2P application is likely to employ multiple security modules. In the next section, we address this particular issue describing two real scenarios.

We have also measured the network usage overhead imposed by P2PSL. Results indicate that the number of bytes added to each message is independent of its size. The overhead for modules Logging, PgpSignature, SymmetricEncryption, and PgpEncryption are, respectively, 172, 252, 315, and 436 B.

8. P2PSL usage by real world applications

This section introduces two case studies, which have been instantiated to evaluate the feasibility of integrating the security layer into real applications. The first consists of a secure file sharing application and the second, of a protected VoIP infrastructure.

8.1. Secure P2P file sharing

File sharing is the most popular application for peer-to-peer systems. Despite the controversy generated by them, they can be a powerful tool for data sharing inside corporations, specially where teamwork is spread through different locations [3]. However, these applications – when applied in a corporate context to assist collaboration – can be exposed to security threats such as improper creation of access channels to sensitive data due to system misconfiguration and virus dissemination through sharing of infected files.

For these applications to be used for corporate activities, a number of issues must be addressed. First, strong authentication and authorization mechanisms need to be applied in order to enforce access rights to data and avoid situations where sensitive information is exposed to non-authorized parties. Second, supposing this data are going to be retrieved on demand through unsecured channels like the Internet, it is very important to guarantee that this data will be neither intercepted nor modified while in transit. These requirements can be satisfied through confidentiality and integrity mechanisms.
Taking into account the previously mentioned concerns, Fig. 9 shows the accumulated overhead P2PSL generates for typical combinations of modules. The measurements include the overhead of the layer itself, plus the modules listed for each line of the plot. PgpEncryption and PgpSignature modules enrich a file sharing application with authentication, confidentiality, and integrity. On the other hand, it is not without cost: such a configuration creates substantial processing overhead, increasing the overall file download time. The overhead stems from the use of asymmetric cryptography. Hence, a simple alternative to reduce this overhead is to employ the SymmetricEncryption module instead of the PGP-based one. Besides, if only file content integrity is to be assured, a lighter configuration can be used.

Fig. 10 shows the total overhead (in seconds) to transfer a 10 MB file. Notice that the overhead generated can vary substantially according to the message size: it ranges from 25 to 1500 s. Although large, these processing delays introduce inherent costs that have been long associated with implementing security. The above study highlights the importance of flexibility and autonomy in choosing which modules a peer will employ. It is also relevant to point out that the choice of modules for a specific peer will be limited not only by the existing module implementations, but also by the processing capacity available for the P2P application in the node.

We have also performed experiments in a Wide Area Network (with round-trip time in the order of 200 ms) to assess resource consumption. Fig. 11 shows the average CPU usage results for transmission (a) and reception (b) of messages. The x-axis represents message size, in log scale, while the y-axis denotes average CPU usage throughout the experiment. In the plot that shows the transmission of messages, one can observe that all curves exhibit a similar behavior: the CPU is busy at 80–97% for messages up to 1024 B long. This is due to the fact that, for smaller messages, a larger time fraction is consumed preparing and dispatching data through the TCP/IP stack. Hence, the sending rates in messages/s tend to be higher. For messages larger than 2048 B, the average CPU consumption is reduced in all cases, except for the PGP cryptography module. For this particular module, the CPU load decreases only when messages larger than 4096 B are used.

In a first glance, one may find surprising that results derived from combinations of two or more modules do not represent the sum of their individual results. For example, the curves showing combinations PgpSignature + Logging and PgpSignature + PgpEncryption + Logging present similar CPU usage, specially for messages up to 1024 B long. This is due to the elastic nature of the sender, which for each combination of modules achieves different maximum sending rates. In this example, for messages of 1024 B, the sender transmits 27 messages/s for the combination PgpSignature + Logging and 10 messages/s for PgpSignature + PgpEncryption + Logging.

Notice now the best and worst cases. With no security modules (Empty Layer), the average CPU load decreases from 83% down to 17% for message sizes ranging from 128 B to 64 KiB (for sending rates of 50 and 3 messages/s, respectively). On the other hand, the worst case is achieved when the modules PgpSignature, PgpEncryption, and Logging are employed together, in which case the average
CPU usage decreases from 96% down to 40% (10 and 3 messages/s, respectively).

Comparing the average CPU usage of sender (a) and receiver (b), observe that the latter consumes approximately 50% less CPU than the former. For the combination $\text{PgpSignature} + \text{PgpEncryption} + \text{Logging}$ with messages of up to 4096 B, the CPU load remains below 40%. From this point on, the load decays smoothly down to around 15%.

In Fig. 11b, one can observe two groups of curves for messages of up to 2048 B. In the upper part, there are four curves, which correspond to combinations that include either $\text{PgpSignature}$ or $\text{PgpEncryption}$ modules. It is possible to infer, therefore, that these modules are, to some extent, responsible for the CPU overload (depending on the message size used). In the lower part of the plot one can see other four curves representing $\text{Empty Layer}$, $\text{Logging}$, and/or $\text{SymmetricEncryption}$. In these cases, the CPU load stays always below 30%, freeing processing power for other system tasks.

Lastly, we also evaluated memory consumption of P2PSL for the same combinations of modules. The obtained results are similar to those observed in non-elastic applications and will be reported in the next subsection.

8.2. Secure Voice-over-IP

In the recent years, Voice-over-IP applications have gone from an early technology aimed at making free phone calls to an enterprise communication solution. These applications are a natural target of attacks such as phone call interceptions and, therefore, depend on confidentiality mechanisms in order to be protected against eavesdropping.

While security should be a fundamental “building block” for any corporate VoIP solution, it is necessary to respect more stringent timely requirements typically found in VoIP applications. These applications are characterized by the transmission of a high number of small messages (e.g., 85 messages/s with voice message size between 40 and 120 bytes [36]). In this particular context, it is mandatory to find the best compromise between security mechanisms and message transmission rates in order to avoid stream interruption or instability due to delays in message delivery.

Fig. 12 shows the message rate (in messages/s) achievable in the current implementation of
P2PSL, for the same combinations of modules used in the previous case study. It is clear that to successfully defend a peer-to-peer VoIP application against eavesdropping, the SymmetricEncryption module should be used. With messages smaller than 4 KiB, it is possible to transmit approximately 70 messages/s, which seems reasonable.

To evaluate resource consumption of this application, we performed a set of experiments in which a sender transmits a fixed-rate stream, using the same combinations of modules employed previously. The stream is characterized by messages of size 128 B sent according to an inter-packet gap of 15 ms (approximately 66 messages/s).

The results measured are shown in Table 1. First notice that the worst case leads to a CPU load of around 80.5%, for modules PgpSignature + PgpEncryption + Logging. In line with findings presented in Section 8.1, symmetric encryption presents an affordable cost, of around 29%. In contrast, the cost associated with PGP encryption is high – increasing the CPU load in around 30%.

As expected, memory consumption is proportionally affected by the number of modules used. However, clearly the amount of resources allocated is negligible taking into account the memory capacity of current computer systems. In the worst case, when modules PgpSignature + PgpEncryption + Logging are used together, P2PSL requires around 127 KiB. These results are also valid for the elastic application discussed in the previous subsection.

### Table 1

<table>
<thead>
<tr>
<th>Modules</th>
<th>CPU (%)</th>
<th>Memory (KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SymmetricEncryption</td>
<td>29.0</td>
<td>43</td>
</tr>
<tr>
<td>Logging</td>
<td>36.0</td>
<td>53</td>
</tr>
<tr>
<td>SymmetricEncryption + Logging</td>
<td>38.0</td>
<td>57</td>
</tr>
<tr>
<td>PgpSignature</td>
<td>49.0</td>
<td>53</td>
</tr>
<tr>
<td>PgpSignature + Logging</td>
<td>51.0</td>
<td>64</td>
</tr>
<tr>
<td>PgpSignature + PgpEncryption</td>
<td>80.0</td>
<td>104</td>
</tr>
<tr>
<td>PgpSignature + PgpEncryption + Logging</td>
<td>80.5</td>
<td>127</td>
</tr>
</tbody>
</table>

P2PSL can offer a limited kind of protection against the latter: the authentication module, along with peer profiles, may be used to deny communication with peers performing such application-level attacks, as well as prevent the situations where peers announce a given content but then refuse to provide it. The slow node attack consists in tampering with announcement messages so that a slow peer is reported as being powerful; all peers resort to such peer, overwhelming it with requests. As a defense, the integrity and authentication modules can be combined with profiling so that (a) messages cannot be tampered with; (b) false reports are associated with malicious nodes; and (c) these can be denied further interaction.

The second type of attack regards confidentiality: when a malicious peer “eavesdrops” supposedly private communication between peers. P2PSL provides protection against confidentiality attacks by means of cryptography modules, allowing for instance messages to be ciphered with the public key of the destination peer. Such key must have been obtained in advance by the sending peer.

Authentication is required so that a peer validates the identity of another peer. Related attacks in P2P networks are hard to defend without help from a centralized entity. According to [11], robust authentication requires a centralized authority. The main authentication attack in P2P is known as the Sybil attack: a malicious peer creates multiple virtual identities for itself. Assuming multiple ids simultaneously or at different times allows a peer to launch a multitude of attacks, such as to reputation (see below). A Sybil is difficult, however, possible when a Certification Authority is used, as long as a physical presence or an external id is not required. Despite providing authentication, P2PSL makes harder, but not impossible, to suffer a successful Sybil attack.

Trust and reputation management is important for certain kinds of applications. For example, they are common in e-Commerce and may be used in file/resource sharing applications to provide incentives for collaboration. Applications that include a reputation system are subject to several kinds of attacks, including traitor and group shilling [37].

One of the most common kinds of attacks is the Denial-of-Service, or DoS. They can be of two types, according to the layer they target – at network layer, they correspond to conventional DoS attacks. At application layer, the P2P network is flooded with spurious requests by malicious nodes, overloading correct nodes that attempt to process requests. P2PSL can offer a limited kind of protection against the latter: the authentication module, along with peer profiles, may be used to deny communication with peers performing such application-level attacks, as well as prevent the situations where peers announce a given content but then refuse to provide it. The slow node attack consists in tampering with announcement messages so that a slow peer is reported as being powerful; all peers resort to such peer, overwhelming it with requests. As a defense, the integrity and authentication modules can be combined with profiling so that (a) messages cannot be tampered with; (b) false reports are associated with malicious nodes; and (c) these can be denied further interaction.

The second type of attack regards confidentiality: when a malicious peer “eavesdrops” supposedly private communication between peers. P2PSL provides protection against confidentiality attacks by means of cryptography modules, allowing for instance messages to be ciphered with the public key of the destination peer. Such key must have been obtained in advance by the sending peer.

Authentication is required so that a peer validates the identity of another peer. Related attacks in P2P networks are hard to defend without help from a centralized entity. According to [11], robust authentication requires a centralized authority. The main authentication attack in P2P is known as the Sybil attack: a malicious peer creates multiple virtual identities for itself. Assuming multiple ids simultaneously or at different times allows a peer to launch a multitude of attacks, such as to reputation (see below). A Sybil is difficult, however, possible when a Certification Authority is used, as long as a physical presence or an external id is not required. Despite providing authentication, P2PSL makes harder, but not impossible, to suffer a successful Sybil attack.

Trust and reputation management is important for certain kinds of applications. For example, they are common in e-Commerce and may be used in file/resource sharing applications to provide incentives for collaboration. Applications that include a reputation system are subject to several kinds of attacks, including traitor and group shilling [37].
to harm a reputation of a correct node or to falsely improve the reputation of a malicious one. Although P2PSL does not include a distributed reputation management mechanism, the security profiling scheme allows a peer to categorize other peers with respect to security according to their past interactions. The requirements associated with a given profile could be augmented with demands reflecting for example the unwillingness of a peer to collaborate. Hence, a distributed reputation mechanism that suits the application might be built on top of the profiling mechanism.

For more complex P2P applications, like some grid computing, an authorization scheme is often necessary. Access to machines or network resources in the P2P network needs to be secure and regulated. A peer may autonomously define access control policies to be enforced locally. As mentioned in Section 4.1, the P2PSL implementation provides an RBAC-based authorization module, which allows users to specify detailed access policies.

Integrity refers to contents of both stored data (e.g., files) as well exchanged messages. Attacks may attempt to tamper with data transmitted in messages from one peer to another. In a file sharing system, a malicious peer may announce and provide corrupted versions of content (or blocks therein) – this is called content pollution. To prevent attacks against integrity, P2PSL allows messages to be digitally signed and checked upon reception.

Last but not least, anonymity and deniability are important aspects as well. Corresponding attacks aim at either exposing information or peer identity, or holding an user legally responsible for some data. As mentioned in Section 2, there are P2P systems like Freenet whose main goal is anonymous information sharing and deniability. P2PSL does not provide explicit support to achieve neither one. On the other hand, it does not prevent the use of anonymizing mechanisms (like Onion routing). Note that anonymity is a conflicting goal with some other P2PSL security features.

10. Concluding remarks

The diversification and dissemination of peer-to-peer applications, specially in scenarios where extensive security requirements must be satisfied (e.g., enterprise content sharing and distributed computing), depends on the availability of flexible approaches to configure and deploy security mechanisms. As mentioned along the paper, existing approaches lack flexibility since they do not provide a wide range of requirements in an integrated fashion. Besides, they demand from the user/application the manipulation of a complex programming interface and the handling of an awkward configuration process.

To address the mentioned earlier issues we have proposed P2PSL (P2P Security Layer). It allows the inclusion of security functionality into P2P applications, respecting the issues of: (i) integration of security aspects into a single application; (ii) isolation between the security mechanisms and both the application and underlying middleware; (iii) asymmetry of security allowing each peer to choose, independently from each other, which requirements should be respected; and (iv) gradual deployment of the scheme in the P2P network. P2PSL has been successfully used in a P2P-based grid computing infrastructure [30]. In addition, we have estimated the feasibility of incorporating the layer into file sharing and streaming applications.

In the future, we expect to perform the incorporation of P2PSL into additional peer-to-peer applications. This will enable us to better evaluate the architecture developed, specially its generality and adherence to other applications. We also intend to develop extra security modules comprising additional security requirements, broadening the applicability of P2PSL.

References


Luciano Paschoal Gaspary is currently an associate professor in the Institute of Informatics at the Federal University of Rio Grande do Sul, Brazil. He received a Ph.D. degree in Computer Science from the same university in 2002. Recently, he has served as general co-chair of LANOMS 2005, the 4th Latin American Network Operations and Management Symposium. He is author of several papers related to network management and security. See http://www.inf.ufrgs.br/~paschoal for further details.
Marinho P. Barcellos is currently a full-time professor in the Postgraduate Program on Applied Computing at UNISINOS University, Brazil. He received a Ph.D. degree from University of Newcastle upon Tyne (UK) in 1998. In 2003–2004, he worked in a joint project between University of Manchester and British Telecomm Research Labs. His research interests include peer-to-peer, multicast, fault tolerance, security, and performance evaluation. Author of several papers in these areas, he is the chair of SBC WP2P 2007, the 3rd. Workshop on Peer-to-Peer Networks. See http://inf.unisinos.br/~marinho for further details.

André Detsch received his B.Sc. and M.Sc. degrees in Computer Science from UNISINOS University in 2003 and 2005, respectively. He is a co-author of Gobolinux, and currently works for IBM.

Rodolfo S. Antunes works as a research assistant, being funded by CNPq Brazilian Research Council. He is a mature undergraduate in Computer Science.