Evaluation of Authentication Mechanisms for Mobile Agents on top of Java

Leila Ismail
College of Information Technology
United Arab Emirates University,
P.O.Box 17555, United Arab Emirates
leila@uaeu.ac.ae

Abstract

This paper presents an evaluation of authentication mechanisms for mobile agents on top of a Java environment. In these mechanisms, the authentication of mobile agents is controlled by the mobile-agents platform using digital signature and a Public Key Infrastructure. An evaluation of these mechanisms has been conducted. The measurements give the cost of the basic mechanisms involved in the implementation of authentication mechanisms integrated within a mobile-agent based platform.

A prototype of a mobile agent system and the authentication mechanisms has been implemented. The implementation experiments have shown the feasibility and the advantage of these mechanisms. The measurements results show insignificant performance overhead of the mechanisms. This paper presents our implementation, a performance analysis and the lessons learned.

1. Introduction

Mobile-agents paradigm is an emergent technology to build distributed computing over the Internet [1] [2] [3]. A mobile agent is a process with its code and data that can migrate in the network from one node to another to perform specific task. Mobile agents can meet and interact with other agents while migrating in the network. The creation, migration, disposal and communication facilities between mobile agents are offered by a distributed middleware called the mobile-agents platform [4] [5].

Security is an important aspect of mobile-agents based distributed computing. The interest to build mobile agents over the Internet has made security an urgent issue to solve. The Internet connection should not be a trap door for the local host. In particular, due to the migration nature of the mobile agent, the latter is assimilated to a virus roaming the network. Hosts wishing to receive mobile agents need to have the assurance that the agent is not a malicious agent.

Java is probably the best known runtime environment which provides facilities for implementing mobile-code based applications and protection mechanisms. The Java compiler generates a bytecode which is interpreted by the Java virtual machine, thus enabling code transfer between heterogeneous hosts. From protection perspectives, the main advantage of Java is the implementation of a sandbox [6] which limits the instructions and consequently the resources used by a mobile code. The Java sandbox is responsible for protecting a number of resources at a number of levels: memory, file system and disk. The memory is protected because the java language is type-safe, which means that the Java language does not allow the use of virtual addresses. The file system and the disk are protected through the use of an access control mechanism based on a policy file. However, in order for agents and servers to control the access to the system resources in a flexible way, an authentication mechanism is needed. Agents and servers may want to implement different security policies based on the authentication of other agents.

Security architectures for mobile agents based on authentication mechanisms [7] have been addressed in the literature [8] [9]. Other authentication experiments have been done in a specific environment made of active networks [10].

In this paper, we propose an experiment of authentication mechanisms, based on digital signature [11] [12] [13] and Public Key Infrastructure (PKI) [14]. We have also conducted an evaluation of the digital-signature based mechanisms used in our implementation. Our model authenticates the sender of the mobile agent. If the sender of the mobile agent is trusted, then the mobile agent is trusted. Mutually suspicious agents will control the access to their resources based on authentication. Our mechanisms have the following advantages:

- Transparency. The authentication mechanisms are implemented within the mobile-agents platform transportation layer. They are completely transparent to mobile-agents applications.
- Modularity. The implementation of these authentication mechanisms is separate from the agents' applications code.
- Portability. The implementation of the authentication mechanisms uses no features specific to Java. Therefore, we believe that it can be ported to any mobile agent-based platform.
- Non-repudiation. The authentication mechanisms are based on digital signature. The digital signature provides assurance of origin or delivery of objects exchanged between a sender and a receiver. Therefore, the sender of the mobile agent cannot deny sending the agent.
- Performance. Our basic measurements show no significant performance overhead due to the use of authentication methods within the mobile-agent based platform. We believe that these results are promising as protection overhead has been always considered as one the burdens to the adoption of protection features.

The authentication mechanisms have been prototyped on top of Java and integrated into a simple mobile agent-based distributed system that we have implemented and that we know the implementations details. This will allow us to perform the
measurements of the basic functions used for authentication in a basic mobile-agent platform.

The rest of the paper is structured as follows. Section 2 presents the related works. In section 3, we present the requirements for the authentication mechanisms in a mobile-agents platform. An overview of the java features that are relevant to our experiment are provided in section 4. Our authentication mechanisms are presented in section 5 and their implementation and the measurements of the Java mechanisms involved are described in section 6. We conclude in section 7.

2. Related Works

The research in the area of mobile agents programming is very active. Several projects have developed execution environments for mobile agents. However, the authentication mechanisms have been partially addressed by some of these environments Telescript [15], D'Agents [16], and AgletIbm [17].

In Telescript, an agent is authenticated using its authority. An authority is associated to an agent on its creation. It is an encrypted message that is associated to an agent. In the public version of Telescript that we have tested, the encrypted message is generated automatically by the telescript engine. However, it can also be generated by an RSA. Telescript uses similar authentication mechanisms to ours. The agent has to present to the destination agent server its authority and proves that it knows its private key that has been used to generate the authority. However, there are no reported experiments with technical demonstrations of the associated mechanisms and their effectiveness and usability within the system.

In D'Agents [18], the authentication is based on PGP (Pretty Good Privacy). However, the digital signature can be also used in the following way. On creation, mobile agents should register to a running agent server. Before sending a registration request agent_begin, the agent encrypts that request with its private key, and then encrypts the result with the agent server public key. On reception of this registration message, the agent server can authenticate the agent. Compared to our approach, PGP is known for its performance drawback. In addition, D'Agents does not include indications about digital certificates and keys distribution and manipulation.

AgletIbm provides basic authentication approach where agents are separated into 2 types: trusted and untrusted. For an agent server\(^1\), trusted agents are those which are dispatched from agent servers within the same domain of the receiver agent servers are within the same domain if they share a secret key. Compared to our approach, there must be a setup of the domain before any exchange between agent servers. In addition, each user of an agent server within a domain must obtain this key signed with the user password. This approach has scalability and usability problems, whereas each new connecting agent server over the Internet must communicate with a system administrator and obtain the secret key to join the domain and that each user must obtain its encrypted secret key. In our approach, we rely on the PKI to distribute the digital certificates without the intervention of the agent servers or the mobile agents. In addition the authentication is done transparently by the agent servers in a transparent way to the mobile agent's application code.

3. Motivations and Requirements

In this section, we present the requirements that we were looking for when designing our authentication mechanisms:

- Sandboxing. Sandboxes refers to agents' confinement. When an agent is hosted by a server, the server must assure that the agent only has access to its own objects or objects for which it has been granted access to by the server and/or by other agents running on that server. The agent must not bypass this rule and directly address objects without the permission of its owner.

- Authentication. Authentication refers to the ability to associate an identity with each agent and to verify that this identity belongs to the one claiming to own this identity. The identity of an agent may be the identity of its creator, or its users or its group of users. It also can be the identity of its dispatcher. An agent must not be able to masquerade another agent by stealing its identity.

- Access Control. Access Control is the definition of the policy that determines the rights given by an agent to other agents.

The first problem is not an open issue since Java implements the minimum level of sandboxing through the use of a strongly typed language which controls agents' accesses to memory. Also Java limits accesses to resources through the use of access policy. The problem of access control is not the issue we focused on, although we believe that any access control mechanism can be easily integrated within our authentication mechanisms. In this paper, we focused on providing agents platforms and agents running on those platforms with the ability to communicate with authenticated agents. The visiting server and/or other agents running on that server may wish to implement different access control policies for different agents based on authentication.

When designing the authentication mechanisms for mobile-agents platforms, we had the following requirements:

- Transparency. The authentication mechanisms should be transparent to agents' applications. Transparency enhances the usability issues of nowadays user-level complex security solutions.

- Usability. The authentication mechanisms should not be too much time-consuming or difficult to use. This decreases the productivity of the users who want to use protection and the protection mechanisms end up by being abandoned or used haphazardly introducing security holes to systems.

- Assurance. The authentication mechanisms should meet the non-repudiation security requirements. The sender and the receiver of the mobile agents should not be able to deny sending or receiving agents. Digital signature offers non-repudiation services to both senders and receivers of mobile agents.

\(^1\) An agent server is a virtual machine within which an agent executes
• Modularity. The agent's application code should not include authentication features. Therefore, the programmer would concentrate on the application programming logic rather than on protection issues.

• Portability. The authentication mechanisms should be general and applicable to all mobile agents systems. The mechanisms should not include features that are specific to Java, but instead only those features that can be ported to any mobile agents system. However, in the rest of the paper, we will describe this model using Java's abstractions in order to be more practical and precise.

4. Java Characteristics

Java [19] is a C++ like object-oriented programming language which is used to generate programs that can be executed on a portable runtime environment that runs on almost every machine type. Since Java is very popular and well known, we will only state the features used in our experiment.

• Code mobility. A key feature of Java is code mobility. Java allows classes to be dynamically loaded from remote nodes.

• Object serialization and de-serialization. Serialization allows instances to be exchanged between different runtimes. Deserialization is the reverse process of serialization. A graph of objects can be reconstructed from the byte stream which includes the serialized graph [20].

• Polymorphism. Refers to the ability to define Interfaces (or types) and classes separately. It is possible to declare a variable whose type is an interface and which can reference objects from different classes that implement the same interface.

• Dynamic binding. It is crucial to mobile code. It means the ability to determine at run-time the code to be executed for a method invocation.

• Digital signature. An essential technique that we used in our authentication mechanisms is digital signature [21]. A digital signature means the ability to apply technical processes on a specific object to prove the object's originator and its integrity. It is based on private/public keys cryptography. A digital signature provides the service of non-repudiation i.e. the inability to deny that one's cryptographic key was used to produce the digital signature. Java provides mechanisms for digitally signing an object using a Digital Signature Algorithm (DSA) as for digital signature [22], and using SHA-1 [23] for hashing. In particular, a pair of private/public key can be generated. An object can be signed and the signature sent along with the object can be verified.

The java language provides the Java Cryptography Extension [19], an implementation for encryption and digital signature.

• Keys and Digital Certificates. The Java environment provides a feature for the management of keys and their associated X509 digital certificates [19]. In particular, it provides an interface to a database called a keystore used to manage a repository of private keys and their associated X.509 digital certificates. The keystore can also be managed through a Java utility called keytool [24].

5. Authentication Model

In mobile agents systems, agents must present proper identities to the agent server that receives them. The agent server will have then the assurance that the received agent is not malicious and can be trusted. When agents are authenticated, then different access rights can be granted to different agents by the servers and other agents exporting services within the agent server. Therefore, authentication is an essential step toward the protection of the agent server resources including other agents and servers running within the agent server.

The authentication model that we have implemented and integrated within our mobile-agents platform is based on digital signature [11] [12] [13]. The digital signature is an electronic signature that is used to authenticate the identity of an object sender and to ensure that the object has not been modified since it has been signed. The digital signature involves the public key cryptography which relies on a pair of keys (a public and a private key) associated with an entity. The advantage of digital signatures is that they allow the non-repudiation service which is one of our objectives.

![Figure 3. Architectural Overview](image-url)

Figure 3 shows an architectural overview of a mobile agent scenario in our implementation. Each agent server in the system communicates (1) with a trusted third party certification authority (CA) to obtain a private key and its corresponding digital certificate. The agent server's digital certificate is digitally signed by the CA. A keystore is associated with each agent server in the system. The keystore is used to store and manage private keys along with their corresponding digital certificates.

When an agent calls the method move(node, port), the sender agent server retrieves (2) its private key and the corresponding digital certificate from the keystore to sign the mobile agent. The signed mobile agent is then sent to a destination agent server (3), which is identified by its IP address node and its port number port. On reception of the mobile agent, the agent server retrieves the CA's digital signature for its keystore (4) and verifies the mobile agent's signature.

The receiver agent server then decides whether to accept or reject the reception of the agent based on the level of trust that the receiver has in the sender. Different access rights can then be attributed to different agents according to an access control policy.

Our authentication model is based on our assumption that the level of trust that a receiver agent server attributes for an
incoming agent depends on the level of trust that the receiver agent server had in the sender agent server.

The main intuition behind our approach is that once an agent is executed by an agent server, it could have been corrupted by a malicious agent server. An agent server executing an agent has full control over the agent's code and state. Many research works are being done in the domain of protecting a mobile agent from a malicious agent server [25][26][27]. However, no satisfactory solution is found till the moment we wrote the paper

6. Evaluation of the Mechanisms

For the purpose of our experiment, we have developed the digital signature authentication mechanisms and integrated them within our mobile agent platform that we know the implementation details. This allows us to identify the Java mechanisms involved in the implementation of a secure mobile agent platform and to measure the cost of these mechanisms.

In this section we describe the implementation of our secure mobile agent platform, and identify the main Java mechanisms involved in the implementation of the protection. We then present a quantitative evaluation of the main components of the security mechanisms and the impact of those components on agent execution.

6.1. Implementation

When a mobile agent calls the method move(node, port), then the sender agent server, on which the agent is running, signs the agent digitally. The sender agent server constructs a message which contains the agent (code and state), its signature, and the agent server's digital certificate. The agent state is composed of a set of Java objects which are managed by the agent itself (a graph of objects). The Java serialization mechanisms allow us to transform the agent's graph of objects into a byte array that we store in the message. The Java digital signature mechanisms allow us to generate a signature that we also store in the message. Then the message is transferred via the network to the destination agent server, identified by its IP address node and its port number port.

Our secure mobile agent platform associates a keystore [19] with each agent server in the system. The keystore is used to store and manage private keys along with their corresponding digital certificates. The agent server's private key and its corresponding digital certificate are created by the agent server's administrator by sending requests to a trusted certification authority (CA). We have created manually a keystore for each agent server in the system. The sender agent server's digital certificate is digitally signed by the CA.

The sender agent server of the mobile agent produces a signature object which uses a hash algorithm. The Java Cryptography Extension [28] provides a DSA algorithm using the SHA-1 message digest algorithm specified by the method Signature.getInstance("SHA1withDSA", "SUN"). The signature object is then initialized for encryption by the sender's private key using initSign(pk), and by the agent's state (to be encrypted) using the method update(state), and then this state is encrypted using the sign() method.

On reception of the message by the destination agent server, the Java de-serialization mechanisms are used to recover the objects graph. The Java security mechanisms are used to verify the signature included in the message. In particular, the receiver agent server retrieves the sender's digital certificate from the message, and verifies whether the sender's digital certificate is valid (whether expired or not) and issued by a trusted certification authority (CA) by using checkValidity(), and verify(key) respectively, where key is the public key of the CA. The CA's public key is retrieved from the receiver agent server's keystore. After successful verification of the sender agent server's digital certificate, the latter is used to retrieve the agent server's public key used to verify the sender agent server's signature. To verify the signature, first a signature object is created with a signature algorithm Signature.getInstance("SHA1withDSA", "SUN"), second this object is initialized with the sender agent server's public key initVerify(publicKey), finally this object is used to verify the mobile agent signature using verify(signature), where signature is the mobile agent's signature retrieved from the message sent by the sender agent server.

To summarize the design of our minimal implementation of a secure mobile agent platform, an agent migration to a destination agent server consists in the following steps:

a) Serialization of the agent state.
b) Retrieval of the sender agent server's private key and digital certificate from the local keystore.
c) Creation of an object signature to be used to sign the agent.
d) Initialization of the signature object with the server private key.
e) Updating the signature object using the agent's state for encoding.
f) Production of the signature of the mobile agent using the signature object from step d).
g) Construction of a message which includes the agent's state, code, signature and sender agent server's digital certificate and sending of the above message to the destination agent server using a TCP/IP connection.
h) Destruction of the agent in the origin agent server.
i) Reception of the message in the destination agent server and creation of a new thread for the execution of the agent.
j) De-serialization of the agent's state.
k) Check the validity of the sender's digital signature.
l) Retrieval of the CA's public key from a local keystore.
m) Verification of the sender agent server's digital certificate using the CA's public key from step k)
n) Initialization of the signature object used to verify the agent's signature.
o) Verification of the agent's signature.
p) Run the agent if verification succeeds.

6.2. Experimentation Environment

Since the objective is to measure the impact of protection on the performance of the sender agent server and the receiver agent server, we run our experiments on a local PC (PC1). This would give us an idea about the processing overhead that is needed for the sake of protection. However, for comparison purposes, we made the same measurements using another PC.
(PC2) of different characteristics (more powerful). The two PCs have the following hardware characteristics:

- PC1: X86 IntelPentiumM@1.80GHz, 512MB of RAM.
- PC2: Intel core duo CPU T2500@2.00GHz, 1 GB of RAM.

We used the JDK 1.5.0_09-b01 for all the experiments described in this paper. The security mechanisms implemented within our mobile-agents platform use the Sun Java Cryptography Extension 1.2.2 [28].

Our purpose is to use the protection on less expensive machines that do not necessarily have a big processing power, which motivates our work here.

### 6.3. Components Costs

To evaluate the cost of each elementary protection mechanism within the platform, we decomposed the protection mechanisms within the agent's migration into several components. These components correspond to the main protection steps presented in section 5.1. To perform these measurements, we developed a minimal agent which iterates a ping-pong secure migration between two agent servers on the same machine. The agent's state size is 356 bytes which include the data parts of the agent.

The protection within an agent's migration is composed of the following components:

- Private Key and Digital Certificate retrieval. This phase corresponds to step b) in the description of platform implementation.
- Agent signature initialization. This phase corresponds to step c), d) and e) in the description of platform implementation.
- Agent signature production. This phase corresponds to step f) in the description of platform implementation.
- Digital Certificate validation. This phase corresponds to step k) in the description of platform implementation.
- CA's Digital Certificate and public key retrieval. This phase corresponds to step l) in the description of platform implementation.
- Digital Certificate verification. This phase corresponds to step m) in the description of platform implementation.
- Signature verification initialization. This phase corresponds to step n) in the description of platform implementation.
- Signature verification. This phase corresponds to step o) in the description of platform implementation.

For each component above, we measured the execution time of the component using a minimal agent (Table 1.).

<table>
<thead>
<tr>
<th>Protection Component</th>
<th>PC1(µs)</th>
<th>PC2(µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Key and Digital Certificate retrieval</td>
<td>1.45</td>
<td>1.14</td>
</tr>
<tr>
<td>Agent signature initialization</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>Agent signature production</td>
<td>7.82</td>
<td>7.70</td>
</tr>
<tr>
<td>Digital Certificate validation</td>
<td>0.12</td>
<td>0.11</td>
</tr>
<tr>
<td>CA's Digital Certificate and public key retrieval</td>
<td>0.68</td>
<td>0.59</td>
</tr>
<tr>
<td>Digital Certificate verification</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>Signature initialization</td>
<td>0.41</td>
<td>0.23</td>
</tr>
<tr>
<td>Signature verification</td>
<td>15.93</td>
<td>13.60</td>
</tr>
</tbody>
</table>

### 7. Conclusion and Perspectives

In this paper, we presented an evaluation of authentication mechanisms for mobile agents on top of Java. In order to measure accurately the costs of the mechanisms, we measured the basic Java mechanisms involved in the implementation of authentication mechanisms for mobile agents. We implemented a minimal mobile agent platform which includes the minimal functionality needed to implement authentication mechanisms. We measured the main security components involved in the authentication and showed that few of them have performance drawback due to an agent's state size increase.
Our authentication model is adequate for a mobile agent platform, since an agent has to be signed every time it is sent. By this way the verification of the integrity of the agent and the non-repudiation are met. It is then easier to identify a malicious agent platform that sends a malicious agent. The conducted experiments showed the feasibility and usability of the implemented authentication mechanisms.

The work presented in this paper continues. We intend to extend our platform to implement and evaluate a dynamic key distribution among agent servers and experiment with larger applications. We are currently developing a distributed agenda application using mobile agents. The application aims to organize a meeting between several participants. Therefore a mobile agent migrates from the agent server of the caller to the meeting to the other participant's agents servers to decide on a meeting time. Our main objective is to evaluate the security issues and solutions involved with in real applications under a real distributed environment.

8. References

[17] Sun Microsystems. "JDK 1.5 Documentation". Sun Microsystems, 2004. URL: http://java.sun.com/j2se/1.5.0/docs/