# Role-based Access Control for E-commerce Sea-of-Data Applications

G. Navarro, S. Robles, and J. Borrell

Dept. of Computer Science
Universitat Autònoma de Barcelona
Edifici-Q, 08193 Bellaterra, Spain.
gnavarro@ccd.uab.es, Sergi.Robles@uab.es, Joan.Borrell@uab.es

**Abstract.** Sea-of-Data (SoD) applications (those that need to process huge quantities of distributed data) present specific restrictions, which make mobile agent systems one of the most feasible technologies to implement them. On the other hand mobile agent technologies are in a hot research state, specially concerning security. We present an access control method for mobile agent systems. It is based on Role-based Access Control and trust management to provide a reliable solution for e-commerce SoD applications. It uses SPKI certificates to implement the role system and the delegation of authorization. It is proposed as an extension of the MARISM-A project, a secure mobile agent platform for SoD application. We also show its functionality with an e-commerce SoD medical imaging data application, which is based on a scenario of the IST project INTERPRET.

## 1 Introduction

Sea-of-Data (SoD) applications have this common characteristics. The user is just interested in the outcomes resulting from processing a huge amount of data distributed among internetworked locations. Data never leaves these locations because a number of reasons, including legal requirements (medical images, for instance), bandwidth limitations, or even physical restrictions (data could be acquired on demand). The user is not on-line during all the time required for data processing: this process might take a considerable amount of time, or the user might be connected through a non reliable network link. SoD applications are extremely useful in many areas such as intrusion detection systems, medical image processing or satellite imagery analysis. Furthermore, SoD applications provide a new prospect for electronic commerce. Data (resource) processing can be charged to users establishing a new variant of selling of indirect services.

Restrictions in SoD applications make them hard, if not impossible, to be implemented with traditional technologies. Mobile agents systems appear to be the only feasible solution to implement SoD applications. In these systems, autonomous software entities (agents) can travel through a network of execution platforms (agencies), allowing the code in SoD applications to be borne to the data. After an agent is launched, the initial agency can be off-line. Thus, the user may be disconnected during the execution of the application. It is also possible to parallelize the execution of processes allowing a high degree of scalability. But mobile agents systems are not a panacea. They introduce a new branch of issues concerning security [3].

One of the most important challenges that needs to be solved is the resource access control. We need a lightweight, flexible, and scalable method to control the access to data and resources in general. Traditional methods are normally based on the authentication of global identities (X.509 certificates). They allow to explicitly limit access to a given resource, through attribute certificates or Access Control Lists (ACLs). They also require a Certification Authority (CA) and a centralized control. This approach makes the system closed from the clients point of view. To be a reliable e-commerce application, it needs to provide a good solution to deal with a great variety of clients. It will be desirable for the clients to access the resource without having to be registered in a CA. Clients may be able to have a simpler key management. On the other hand, we will have a reduced and controlled number of agencies. They can afford to have keys with relatively long validity time ranges and be registered in a CA.

An alternative to implement access control are authorization infrastructures. These infrastructures are based on trust management and allow to assign authorizations (permissions or credentials) to entities and delegate trust from one entity to another. One of these infrastructures is the *Simple Public Key Infrastructure* (SPKI) [5], which seems to be one of the most accepted. SPKI provides a good base to implement the access control method. There are security frameworks providing SPKI functionalities [8], an it is probably the most standard solution to implement trust management mechanisms, such as delegation of authorizations. There are also some approaches using authorization infrastructures to implement access control methods [1], [9].

In order to make access control easier, it is also interesting to use an approach similar to Role-based Access Control (RBAC) [11]. Instead of having to provide and/or revoke specific permission to a principal, RBAC allows to determine the privileges of a principal by its role membership.

In this article, we present a solution for e-commerce SoD applications. We adapt RBAC for mobile agents using SPKI. Our model allows to authorize a mobile agent to access a given resource and control its access with quite flexibility. The mobile agent does not need to carry any kind of information with regard to the resource access. This avoids the inconveniences of storing sensitive information in the mobile agent. Our model is going to be implemented as an extension of the MARISM-A platform [14], a secure mobile agents platform. To clarify and explain our proposal, we will explain an example application based on a scenario of the project INTERPRET IST-1999-10310 (*International network for Pattern Recognition of Tumors Using Magnetic Resonance*, `http://www.carbon.uab.es/interpret`).

In Section 2 we give an overview of SPKI. Section 3 introduces the MARISM-A platform and the extension added to support the RBAC model through SPKI. We present our adapted RBAC model in Section 4 and the example SoD application in Section 5. Section 6 shows the functionality of the proposed model over the example application and considerations about the distribution of the model. And finally Section 7 contains our conclusions.

## 2 SPKI

The base to our proposal is SPKI (more formally named SPKI/SDSI) [5]. It is an infrastructure which provides an authorization system based on the delegation of authorizations and a local name model. It provides mainly two kind of certificates, authorization and name certificates. Any individual, software agent or active entity in general is called a *principal*. It is a *key-oriented* system, each principal is represented and may be globally identified by its public key. We can say that in SPKI a principal is its public key. Since it does not need a certification authority, each principal can generate and manage its keys. A key is a generic cryptographic key pair (public and private). Currently the SPKI specification supports RSA and DSA keys. The representation format used by SPKI is S-expressions [13].

An authorization certificate has the following fields:

- Issuer (*I*): principal granting the authorization.
- Subject (*S*): principal receiving the authorization.
- Authorization tag (*tag*): specific authorization granted by the certificate.
- Delegation bit (*p*): if it is active, the subject may forward delegate the authorization received.
- Validity specification (*V*): specifies the validity of the certificate through a time range and on-line tests.

It is signed by the issuer. The on-line tests from the validity specification field, provide the possibility of checking, at verification time, the validity or revocation of the certificate. We will use the 5-tuple notation: *(I,S,tag,p,V)*, to denote a SPKI authorization certificate.

In a normal situation there will be a principal controlling a resource, which delegates an authorization. The authorization may be further delegated to other principals. If a principal wants to access the resource, it needs to provide an *authorization proof*. Te proof is a certificate chain, which binds the principal controlling the resource to the one requesting the access. To find this certificate chain there is a deterministic algorithm. *Certificate Chain Discovery Algorithm*[4], which finds the authorization proof in polynomial time.

In SPKI a principal may have a local name space and define local names to refer to other principals. To define a name, a principal issues a name certificate. It has an issuer, subject, validity specification, (just as an authorization certificate) and a name. The *issuer* defines the *name* to be equivalent to the *subject*. For example a principal with public key $K$ may define the name $Alice$ to be equivalent to the the principal with public key $K_A$. Now $K$ can refer to the principal $K_A$ by the name $Alice$ instead of the public key. Such a name certificate can be denoted as:

$$K\ Alice \longrightarrow K_A$$

meaning that $K$ defines the name $Alice$ in its local name space to be equivalent to $K_A$. If a principal wants to refer to a name defined in another name space, it just has to add the local name space owner's public key to the name as a prefix. When we say $K\ Alice$, we mean the name $Alice$ defined in $K$'s local name space.

SPKI also provides the ability of defining compound names. Names that refer to other names which may also reference other names and so on. For example, the principal $K_B$ can define the following name in its local name space:

$$K_B \; employee \; \longrightarrow \; K \; Alice$$

It defines the name *employee* to be equivalent to the name *Alice* defined in $K$'s local name space. Note that it is referring to $K_A$ without knowing it.

This is a key concept in our proposal since we will consider a role as a SPKI local name.

## 3 MARISM-A

As said before the proposed access control model is an extension of our MARISM-A platform [14] (`http://www.marism-a.org`). MARISM-A is a secure mobile agent platform implemented in Java, it is implemented on top of the JADE-LEAP system (`http://sharon.cselt.it/projects/jade`, `http://leap.crmparis.com`), which follows the standards proposed by FIPA [6].

The basic element of the MARISM-A platform is the agency, the environment for the execution of agents. An agency consists of a directory service, an agent manager, and a message transport service. An agent system has several agencies distributed on a network. Each agency in controlled by an entity (its owner).

Agents are software units executing in the agencies on behalf of their owners. Agents in MARISM-A can be mobile or static, depending on the need of the agent to visit other agencies to fulfill its task. There are several types of mobile agents according to the characteristics of its architecture: basic or recursive structure, plain or encrypted, itinerary representation method, etc. Agents can communicate each other through the agency communication service.

One of the novel aspects introduced in the MARISM-A platform is the flexibility of the agent architecture. Different security solutions have some especial agent requirements. Instead of focusing on a specific type of agent, there are different agent architectures. Some security mechanisms are applicable only for certain types of agents. Even mobility is a feature of only some agent architectures. Moreover, our design allows to have several types of agents living together in a heterogeneous environment.

Most bibliographic references on agents do not make a clear distinction between different parts of an agent. Some of them suggest the need of considering independent some internal parts, especially for mobile agents. This is the case of agent data in [15], of agent code in [2], or agent itinerary in [10]. The independence of these parts plays an important role for some agent protection mechanisms, whereas it is unnecessary for others. In MARISM-A, the architecture of the agent is an adaptable model that determines the different parts in which an agent is divided and the combination of security, integrity, or other mechanisms included in it.

All mobile agent architectures share some basic aspects, such as the differentiation of internal parts and migration mechanisms. A mobile agent consists of code, data, state, and an explicit itinerary. Code is the set of instruction describing the execution of the agent. Data is an information storage area that can be used by the agent at any moment for reading and writing and goes with it all the time. Results of executions are stored in this area, normally using some convenient protection mechanisms. State is like the data part of the agent but reserved to store the agent information related with its state. Explicit itinerary is a structure containing all agencies that are going to be visited by the agent on its life cycle. Itineraries consist of several basic structures: sequences, sets, and alternatives. These structures can be combined to build complex itineraries. In a sequence, the agent will migrate to each agency one after the other. In a set, a group of agencies will be visited by the agent in no special order. On the other hand, only one agency of those listed in an alternative will be visited by an agent, depending on some conditions.

MARISM-A considers a minimal security infrastructure to protect the communications between agencies. All the agencies are registered in a CA, and we use SSL to provide both confidentiality and authentication for agency communications.

It is important to assume that agencies untrust each other. Therefore, they might try to modify results carried by the agent, or to gain knowledge about its itinerary, to favor themselves to the detriment of the rest. It is also reasonable to assume that agencies are not malicious and they do not seek to adversely affect the owner of the agent (the client), or the agent itself.

From now on we will use the following notation:

- $E_i(m)$: encryption of $m$ using a symmetric cipher with $i$'s secret key.
- $P_i(m)$: encryption of $m$ using an asymmetric cipher with $i$'s public key.
- $S_i(m)$: signature of $m$ using $i$'s private key.
- $hash(m)$: hash function of $m$.
- $hash_{K_i}(m)$: keyed hash function of $m$ using $i$'s secret key.

Subsections 3.1 and 3.2 introduce the architecture of MARISM-A static agents and the mobile agents with explicit itinerary as an extension to MARISM-A mobile agents.

### 3.1 Static Agents

A MARISM-A static agent has the following form:

Agent = ControlCode, State, Code, Data

Because agent management code is in the agent itself, it is indifferent for the platform to deal with mobile or static agents. There are not many words to say about security in static agents. Communication and interface with other agents are provided by secure services of the agency. Data protection is assured by the agency too, and there is no itinerary to protect here.

### 3.2 Mobile Agents with explicit itinerary

Agent code is split into several pieces. There is a main code that will be executed in all agencies (Common Code), and as many code fragments as agencies are in the itinerary, each one to be executed in a particular agency (Local Code). This feature makes MARISM-A very useful in some types of application where execution is context dependent. The agent changes after a migration. This agent aspect dynamism allow several security mechanisms to be applied.

The agent has the following structure:

$$\begin{aligned}\text{Agent}_i = \text{ControlCode, StateData,}\\ \text{CommonCode, GlobalData, ResultsData,}\\ (\text{LocalCode}_0, \text{LocalData}_0, \text{Agencies}_1), \ldots,\\ (\text{LocalCode}_{n-1}, \text{LocalData}_1, \text{Agencies}_n),\\ (\text{LocalCode}_n, \mathit{Nil})\end{aligned}$$

$\text{Agencies}_i$ is the agency (or agencies, depending on the type of itinerary) the agent is going to visit (migrate) next. The agent that is sent to the next hop of the itinerary ($\text{Agent}_{i+1}$) has the same structure. The last host is identified because of a *Nil* next agent. CommonCode is executed by all agencies when the agent immigrates and before the specific LocalCode. Programming is simplified by using this common code to include the non agency dependent code only once. The control code in the agent deals with the functions of agent management, in this case extracting the relevant parts of the agent.

It might be interesting to protect integrity and secrecy of data that has been written in some agency. In an e-commerce application, for instance, where agencies represent shops and agents act on behalf of buyers, it might be necessary to protect offers from rival shops. The method to provide the secrecy and integrity required for this data in this agent architecture is based on a hash chain. Some of the data area is reserved to store results from executions (Results Data). Results are not stored plain, but they are firstly encrypted using agent's owner cryptographic information. Only the owner of the agent will be able to read these results. Once the result has been written a hash of the Result and previous hashed information is calculated, signed and written also. This hash has information about the identity of next agency in the itinerary, so that no agency can neither modify the result area nor remove some result. Each agency verifies during immigration that all hashes in the Results Data are correct. This is the format of the Results Data area:

$$\begin{aligned}\text{Results Data} = P_o(R_1), S_1(hash(P_o(R_1), Id2)),\\ P_o(R_2), S_2(hash(P_o(R_2), hash(P_o(R_1), Id2), Id3)), \ldots,\\ P_o(R_n), S_n(hash(P_o(R_n), ..., hash(P_o(R_1), Id2), Id0)))\end{aligned}$$

where $P_o(R_i)$ is the encryption of the result of agency $i$ using the public key of the owner of the agent ($\text{PubKey}_o$) and $hash()$ is a hash function.

We also need a way to ensure the agent's integrity. The owner, before sending the agent, computes a keyed hash of the whole agent excluding the Results Data, $hash_{K_o}(initial\_agent)$. Then when the agent finishes its execution, the owner can verify the agent's keyed hash.

To protect the itinerary we use the following encryption schema:

$$
\begin{aligned}
&\text{Agent}_i = \text{PubKey}_o\text{, ControlCode, StateData,}\\
&\quad\text{CommonCode, GlobalData, ResultsData}\\
&\quad E_1(\text{LocalCode}_0\text{, LocalData}_0\text{, Agencies}_1), \ldots,\\
&\quad E_{n-1}(\text{LocalCode}_{n-1}\text{, LocalData}_{n-1}\text{, Agencies}_n),\\
&\quad E_n(\text{LocalCode}_n, \textit{Nil}),\\
&\quad hash_{K_o}(initial\_agent).
\end{aligned}
$$

where $E_i$ is a symmetric encryption function using agency $i$ symmetric key. As we will see the encryption is performed by the agency itself before the whole agent is constructed. So the symmetric key is only used by the agency and it does not need to be distributed.

A variant of this agent is the mixed one, where the list of information for agencies is scrambled. This makes it not possible to know which is the part of the agent that will be executed in a given agency.

## 4 Access Control for SoD applications

One of the first problems we found when planning the authorization model, is if the mobile agents should have a SPKI key and be considered as principals. A mobile agent cannot trivially store a private key, so it cannot perform cryptographic operations such as digital signatures. There are some propositions to store sensitive information (private keys) in mobile agents [12]. But the problem arises when the mobile agent uses the private key to compute a cryptographic operation. The agency where the agent is in execution will be able to see the private key. As a result we consider that a mobile agent should not have a private key.

Our solution is to delegate authorizations directly to the agent. This way the mobile agent does not need to carry any kind of authorization information, making the agent more simple and lightweight. This issue will be discussed in section 6.2.

The main components of the access control method can be seen as independent modules. Each module is implemented as an static agent, has a SPKI key, and it is considered as a SPKI principal. The modules are:

**Authorization Manager (AM)** it manages the delegation of authorizations, issuing SPKI authorization certificates. It follows a *local authorization policy*.

**Role Manager (RM)** it manages the roles (mainly the role membership) by issuing name certificates. It follows a *local role policy*.

**Certificate Repository Manager (CRM)** it manages a certificate repository. Provides services such as certificate chain discovery.

**Resource Manager (DM)** it is an authorization manager, which controls a resource (data), it has to verify resource access requests. Normally its authorization policy will be quite restrictive, delegating to an authorization manager the responsibility of performing complex authorization tasks.

Figure 1 shows a simple schema of the model with two DMs, an AM, a RM, and a CRM. The RM defines the roles and determines its membership. The DMs delegate the authorizations related to the resources to the AM, and the AM delegates authorizations to the roles. Each static agent stores the issued SPKI certificates in the certificate repository through the CRM (denoted by broken lines).

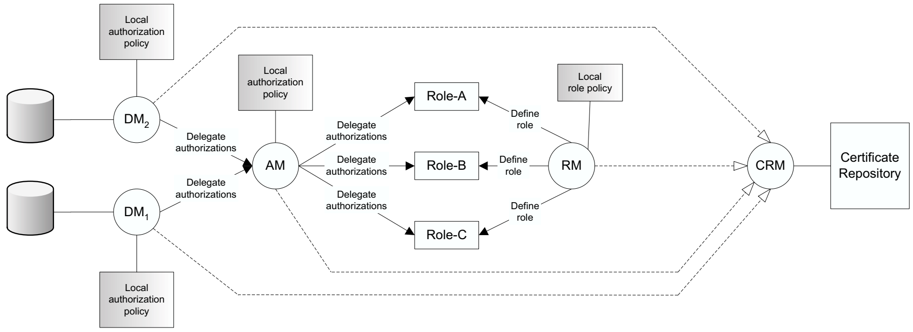


**Fig. 1.** Access control modules

### 4.1 Authorization Manager (AM)

The main functionality of the AM is to provide authorization certificates under request. To obtain an authorization certificate, a principal sends a request to the AM specifying the specific authorization, it wants to obtain. Then the AM decides whether to issue the certificate or not, and under what conditions it has to be issued. To do that, it follows its local authorization policy. Since the policy is local to the AM agent, it does not need to follow any specification and its format is implementation dependent.

We propose an authorization policy, described as a SPKI ACL, where each rule can be expressed as an ACL entry in S-expression format. A SPKI ACL entry is an authorization certificate without the issuer and it does not need to be signed because it is local to the AM and stored in secure memory. It has the following fields:

– Subject: the principal receiving the authorization. It may be a role or another AM.
– Authorization tag: specifies the specific authorization that the subject can obtain. SPKI allows quite flexibility to specify the authorization tag with S-expressions.
– Delegation bit: determines whether the subject may receive the right to delegate the authorization or not.
– Validity specification: allows to limit the authorization to a time range, and include some on-line tests to verify the validity or revocation of the authorization certificate.

To be more specifics, the AM will receive authorization delegation requests from a RM or another AMs. It has to delegate authorizations to roles or to other AM which will finally authorize roles.

### 4.2 Role Manager (RM)

The RM is responsible for assigning and managing roles, and determines the role membership. The use of roles facilitates the access control management and the specification of policies. The main idea is to exploit the advantages of Role Based Access Control (RBAC) [11] and trust management. The RM assigns a role by issuing a SPKI name certificate following its local role policy. It can also assign a role to a role defined by another RM, thus allowing the delegation of role membership management. Section 6.1 details how roles are assigned and used.

Each RM has a local role policy which determines what roles does it manage. It also includes rules to determine if a given principal requesting a role membership has to be granted or not. If we choose to describe the role policy as a SPKI ACL, it is quite similar to an authorization policy. Now the subject of the SPKI ACL entry is a principal or another role and the authorization tag determines the role that the subject can have.

### 4.3 Certificate Repository Manager (CRM)

A CRM implements a certificate repository. For example, one agency may have one CRM to collect all the certificates issued by agents inside the agency. The CRM provides the repository and all the services needed to query, store or retrieve the certificates in the repository. It also provides a certificate chain discovery service. A principal can make a query to the CRM to find a specific certificate chain. This way we solve the problems derived from certificate distribution and leave the task to perform chain discoveries to the CRM and not to the other principals. It decreases the communication traffic, certificates do not need to travel from one principal to another and reduces the task that generic principals need to perform.

### 4.4 Resource Manager (DM)

The main task of a DM is to control the access to a resource (data). It holds the master SPKI key to access the resource, delegates authorizations to AMs, and verifies that an agent requesting access to the resource has a proper authorization.

Another important feature of a DM is to issue *Certificate Result Certificates* (CRC) to agent hashes, see 6.2.

As it has to delegate authorizations issuing authorization certificates it also acts like an AM and follows a local authorization policy. But this policy is much more restricted. A DM only has to issue authorization certificates to AMs and a special certificate to mobile agents (see 6.2), which are quite straightforward operations.

## 5 Example Application

This example is derived from the project IST INTERPRET, which provides a clear example of a SoD application. The example application is going to be developed using the extensions of the MARISM-A platform to support the proposed access control. Consider a medical SoD application for radiology images. There are several hospitals, research centers, and companies with a radiology department which produces some kind of sensitive, and possibly expensive, radiology images such a magnetic resonances or high resolution radiologies. Each center organizes the data in a database accessed by at least one agency with DMs. The application may provide the ability for clients to process the distributed data in a variety of ways, for example testing a classification algorithm. The owner of the data may also provide itself classification services. It may have a trained classification system, which a client may use to classify a reduced set of data provided by herself.

The reason for using a mobile agent approach in this application, is due to the high quantity of distributed data, which is difficult to centralize. Also because it is quite sensitive medical data, which the hospitals are normally not allowed to give it to someone else. That is, a mobile agent processing the data, may get back to the client with the obtained results, but not with the data.

We will consider each participating entity as a principal. A principal may be a static agent or an individual (normally the owner of a mobile agent) with its own SPKI key. We consider three kinds of principals, data producers, data consumers, and process consumers:

– *data producer*: updates the database, adding new images or replacing existing ones.
– *process consumer*: provides a reduced set of data, and wants to use some processing service provided by the agency (normally a complex trained algorithm such as a classification one).
– *data consumer*: it provides a code to be executed with the data provided by the agency.

A simple definition of roles for the example application may be:

– *physician*: authorized as data and process consumer for all the resources.
– *external_physician*: authorized as process consumer for a reduced set of data.
– *radiography_technologist*: authorized as a data provider.
– *external_researcher*: authorized as data consumer for a restricted set of data.

These roles may be hierarchically extended, for example as *radiography_technologist*, there may be *radiographer*, which provides only radiographies and *mr_technologist*, which provides only magnetic resonances. Specially the *external_researcher* role, which may be seen as a client, may have several sub-roles to be able to specify several specific authorizations for different kinds of clients.

## 6 Access Control Management

Given the example application we will show the functionality of the access control method. The main features are the role system and the delegation of authorizations to mobile agents. A principal may be authorized to access a resource as a role member. The AM may give several authorizations to an specific role. Then a principal belonging to that role, has all the authorizations of the role. We already said that we do not consider a mobile agent as a SPKI principal. Thus we need a way to authorize mobile agents and control its access to resources.

We also consider the distribution of the access control management by distributing the modules. We can distribute several modules, or just one, for example. This makes the model easily adaptable to specific applications. Since a module is implemented in a static agent, to distribute a module means to use several static agents, which may operate independently.

### 6.1 Roles

An important issue of the RM is that it is the main responsible to grant access permissions to principals. When a principal requests a role membership and succeeds, it automatically has all the authorizations of the role. In some cases, specially *external_researcher* membership, the RM will need to perform some kind of economic transaction to grant the membership. That is, granting membership will require a special protocol involving a payment processes through, for example, a third party credit card issuer.

Another important issue is that the role membership can be restricted through the validity specification of the name certificate, which grants the membership. That is, it can have a *not-after* and *not-before* time range and some on-line tests [5].

### 6.2 Mobile agent authorization

A client as a principal may be member of a role or roles, say *external_client*. It may be authorized to access resource $A$ with a mobile agent. Since mobile agents cannot have private keys, we can not delegate authorizations to the mobile agent or make it member of a role. Our approach is to delegate the authorization to a hash of the agent. The subject of a SPKI authorization certificate and any SPKI principal in general can be a public key or a hash of a public key. So a hash may be seen as a principal, subject of a certificate. This idea does not really follow the SPKI specifications, since it is not the hash of the public key, it is not a principal. Thus we need to extend the SPKI specifications to introduce this idea.

As we said before the mobile agent is constructed from the itinerary, separately including the code to be executed in each agency. Let $m_i$ be the code to be executed in the agency $i$. The client already has an authorization to access resource $A$, which is controlled by $DM_A$. Once the client has specified all the $m_i$s it constructs the itinerary and proceeds to get the authorization for the agent. The main idea is to request a *Certificate Result Certificate*(CRC) to $DM_A$ having the hash of $m_i$ as the subject. The CRC is an authorization certificate, which resumes a certificate chain, in this case the authorization proof for the client to access resource $A$. The process involves the following steps:

1. The client sends a CRC-request to $DM_A$. It includes the specific authorization it wants to obtain, the code $m_i$, and the client's public key. This request is signed by the client.
2. The $DM_A$ requests the CRM to verify if the client is authorized to access the resource. That is, verifies if there is an authorization proof which allows the client to access the resource.
3. If the authorization is correctly verified, the $DM_A$ computes the hash of the code, and issues an authorization certificate which has $DM_A$ as the issuer and the hash of the code as the subject. The specification tag and the validity specification is the intersection between the ones from the client's CRC-request and the ones returned in the authorization proof request.
4. Finally the $DM_A$ encrypts the code $m_i$ with a symmetric cipher. It uses a private key only known by the $DM_A$. The $DM_A$ is the only one who is able to decrypt $m_i$.

Once the mobile agent is constructed it will be able to access the resource. The mobile agent will travel to the agency and request access to $DM_A$. The $DM_A$ just has to decrypt and compute the hash of the agent code ($m_i$); and check if there is an authorization certificate, which directly authorizes the hash to access. This authorization verification is straightforward, since it does not require the generation of a full authorization proof.

This approach allows to delegate authorizations to mobile agents. Note that the mobile agent does not need to include any kind of authorization information, it just has to provide the specific code so $DM_A$ can compute the hash.

One thing we have not explicitly talked about is how to control the proper behavior of the mobile agents. In our example, how do we know that a mobile agent is not *stealing* data?. First of all, the process of authorizing a mobile agent involves the computing of the hash of the piece of code of the agent, which is going to be executed in the agency. So we can easily log this code for auditing purposes. It is also feasible for an agency to include a local monitoring system looking for anomalies in the behavior of the agents.

### 6.3 Distribution of Role Management

Due to the local names provided by SPKI, the role management can be easily distributed. We can have several RMs managing its local roles and use compound names to reference one local role to another. For example, consider we have two RMs, named $RM_A$ and $RM_B$. Each one has its local roles definitions, $RM_A$ may define:

$$RM_A\ radiography\_technologist \longrightarrow K_1$$
$$RM_A\ physician \longrightarrow K_2$$
$$RM_A\ physician \longrightarrow K_3$$
$$RM_A\ companyB\_client \longrightarrow RM_B\ ext.\_researcher$$

That is, it says that the principal $K_1$ is member of the *radiology_technologist* role; the principals $K_2$ and $K_3$ are members of the role *physician*. And that the name *external_researcher* (which is also a role) defined in the local name space of $RM_B$ is member of the role *companyB_client*. Then $RM_B$ may define:

$$RM_B\ external\_researcher \longrightarrow K_4$$
$$RM_B\ external\_researcher \longrightarrow K_5$$

So the principals $K_4$ and $K_5$ are members of the role *external_researcher* defined by $RM_B$. And they are also members of the role *companyB_client* defined by $RM_A$. Note that each RM defines independent roles, both RMs could define locally two roles with the same name, and they will be considered as different roles by the system. Is important to notice that all the roles, as SPKI names, are local to each RM. We can globally identify the role by adding the public key of the RM as a prefix of the role (just as a SPKI name). This independence of role definitions makes the system easily scalable and distributed. Note that in the example we can say that the role management is distributed over the two RMs since both of them take part in the hole role management. So independent RMs can interact in the same model without having to redefine roles.

This can be also seen as trust management, in some way $RM_A$ trusts $RM_B$ to manage the role $RM_A$ *companyB_client*.

### 6.4 Distribution of Authorization Management

The distribution of the authorization management is achieved by distributing the management over several AMs. This distribution is straightforward. Each AM manages authorizations following its local policy. It can only delegate an authorizations that it has received. To be more precise an AM or any principal may delegate a certificate granting an authorization it does not have. But any principal receiving the authorization will not be able to have the proper authorization proof, since the certificate chain will be broken.

There will be no conflict between several AMs. If there is an authorization proof for one principal to access a resource, the principal will be able to access no matter which AMs or principals have interfered.

### 6.5 Distribution of the Certificate Repository

The distribution of the certificate repository is a complex task. All the authorization proofs are obtained from the repository. In fact it is the CRM, which performs the certificate chain discovery. To distribute the repository will considerably increase the complexity. We need to use a distributed certificate chain discovery algorithm, which adds not only complexity to the implementation but also introduces the need for more communication and process resources.

The application we are implementing does not impose the distribution of the certificate as a must. In fact it can easily be implemented with a centralized repository. So there is no need to add complexity to the system by distributing the repository.

There is some work done in relation to distributed certificate repositories and chain discovery, such as dRBAC [7], which could be used if an specific application really needs to distribute the certificate repository.

## 7 Conclusions and further work

We have proposed an access control model for e-commerce SoD applications, based on a mobile agents platform. It provides a simple, flexible, and scalable way of controlling the access to resources. It takes the advantages of RBAC and trust management ideas. The proposed model is an extension of the MARISM-A project. A secure mobile agents platform for SoD applications. We have also introduced an example application, a medical SoD imaging application based on the IST project INTERPRET.

Our solution provides a secure migration for agents with protected itineraries and we solve the secure resources access control and the authorization management. Even though, there are some problems which still being unsolved, like for example some subtle and limited replay attacks from one agency to another; or open problems like malicious hosts acting against agents. Some existing solution can solve the former problems [16] with a high cost in scalability, distribution, and complexity, but the later ones are still open problems [3].

We are working on the implementation of the proposed model. This process involves the study of additional aspects. For example considering alternatives to implement the local policies. By using SPKI ACLs, the policy is based on SPKI keys. This may be reflected in limitations of the key management. We also want to consider issues such as anonymity, specially relevant in key-oriented systems.


## Acknowledgments

This work has been partially funded by the Spanish Government Commission CICYT, through its grant TIC2000-0232-P4-02, and Catalan Government Department DURSI, with grant 2001SGR 00219.


## References


1. T. Aura. Distributed access-rights management with delegation certificates. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, volume 1603 of *LNCS*, pages 211–235. Springer, 1999.
2. J. Baumann, F. Hohl, K. Rothermel, and M. Straßer. Mole - Concepts of a Mobile Agent System. *Special Issue on Distributed World Wide Web Processing: Applications and Techniques of Web Agents*, 1(3):123–137, 1998.
3. D. Chess. Security issues of mobile agents. In *Mobile Agents*, volume 1477 of *LNCS*, pages 1–12. Springer-Verlang, 1998.
4. D. Clarke, J. Elien, C. Ellison, M. Fredette, A. Morcos, and R. Rivest. Certificate chain discovery in SPKI/SDSI. *Journal of Computer Security*, 9(9):285–322, 2001.
5. C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. RFC 2693: SPKI certificate theory. The Internet Society, September 1999.
6. Foundation for Intelligent Physical Agents. FIPA Specifications, 2000. `http://www.FIPA.org`.
7. E. Freudenthal, T Pesin, L Port, E Keenan, and V Karamcheti. dRBAC: Distributed role-based access control for dynamic coalition environments. New York University, Technical Report TR2001-819.(to appear ICDCS 2002), 2001.
8. Intel Architecture Labs. Intel Common Data Security Architecture. `http://developer.intel.com/ial/security/`.
9. L. Kagal, T. Finn, and A. Joshi. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, pages 154–157, Dec. 2001.
10. G. Karjoth, N. Asokan, and C. Glc. Protecting the Computation of Free-Roaming Agents. In *Proceedings of the Second International Workshop on Mobile Agents*, LNCS 1477, pages 194–207. Springer-Verlag, 1998.
11. D. Rerraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R Chandramouli. Proposed NIST standard for role-based access control. In *ACM Transactions on Information and System Security*, volume 4, pages 224–274, August 2001.
12. J. Riordan and B. Schneier. Environmental key generation towards clueless agents. In *Mobile Agents and Security*, pages 15–24, 1998.
13. R. Rivest. S-expressions. Internet-draft: `<draft-rivest-sexp-00.txt>`. The Internet Society, 1997.
14. S. Robles, J. Mir, and J. Borrell. Marism-a: An architecture for mobile agents with recursive itinerary and secure migration. In *2nd. IW on Security of Mobile Multiagent Systems*, Bologna, 2002.

15. M. Straßer and K. Rothermel. Reliability Concepts for Mobile Agents. *International Journal of Cooperative Information Systems (IJCIS)*, 7(4):355–382, 1998.
16. M. Straßer, K. Rothermel, and C. Maifer. Providing Reliable Agents for Electronic Commerce. In *Trends in Distributed Systems for Electronic Commerce*, LNCS 1402, pages 241–253. Springer-Verlag, 1998.