

# An Authorization and Access Control Scheme for Pervasive Computing

Linda A. Staffans<sup>1</sup> and Titos Saridakis  
NOKIA Research Center  
PO Box 407, FIN-00045, Finland

## ABSTRACT

The existence of a central security authority is too restrictive for pervasive computing environments. Existing distributed security schemes fail in a pervasive computing environment with limited terminals. Better fitted are schemes, that do not rely on the presence of a central security authority, yet allows for the application of a common security policy. This paper presents such a distributed security scheme, where pieces of information of the same sensitivity are grouped together and protected by a pair of private encryption/decryption keys. Users gain access to certain information by obtaining the key pair of the corresponding group. Depending on the security policy applied in a given environment, the keys can be obtained either directly by the security authority which issues the keys or by another user that possesses them. Similarly, depending on the applied security policies, the access to information may require the user to authenticate himself. In the scheme we present, the authentication is based on certificates that users may obtain from the security authority at an unsuspected time prior to the information access.

## KEY WORDS

Distributed Security Scheme, PKI, Security Policies

## 1 Introduction

The explosive evolution of wireless communications in the past decade, combined with the equally impressive progress in the hand-held device technology has opened the way for pervasive computing. Pervasive computing builds on the results from the domains of distributed systems and mobile computing, but presents several new areas of interest such as smart spaces, invisibility, localized scalability, and masking of uneven conditioning, i.e. graceful service quality degradation when space “smartness” decreases [1]. A number of research projects (e.g. Oxygen [2], Aura [3], Endeavour [4], a.o.), each with an impressive list of participants both from the industry and from the academia, have emerged and started tackling issues related to pervasive computing.

Security is a hot topic of research in pervasive com-

puting. The centralized nature of the authentication and access control authority of traditional approaches is not affordable in pervasive computing as has been successfully argued elsewhere [5]. Instead, the confidentiality and integrity of information must be ensured by a distributed scheme where the interacting parties do not rely on the presence of a central security authority. Traditional distributed schemes cannot be applied in a pervasive computing environment, since the terminals used have too restricted computing power and storage space to be able to run authorization and access control operations. Therefore, we need an authorization scheme, which does not require a central security authority, yet frees the terminals from complex computational processes or heavy memory usage.

This paper presents a distributed authorization and access control scheme for a pervasive computing environment. We also describe an implementation of that scheme. The scheme is based on the distributed security scheme for nomadic computing described in [6], but has been enhanced to fit the needs of pervasive computing. The scheme provides support for security policies that guarantee the confidentiality and integrity of information exchanged without the presence of a central security authority.

In our scheme, parties which intend to access information in a pervasive computing environment, gain access to the desired information by obtaining a pair of private keys used to encrypt that information. The keys along with the credentials that, depending on the security policy, might be required for user authentication in a given environment, are received by the user in the form of an *admission ticket*.

The credentials ensure that only authorized entities will be allowed to connect to services available in the corresponding pervasive computing environment. Still, all data communicated within that environment is encrypted with different keys, depending on the access control group a specific piece of data belongs. Different access control groups may contain information of different sensitivity, reflecting different security levels defined in a variety of security policies that may be used in that pervasive computing environment. Only the entities that are allowed to access information that belongs to a certain access control group receive the encryption keys that correspond to that group in their admission tickets.

The remainder of this paper is structured as follows:

---

<sup>1</sup>Linda A. Staffans is currently with the Telecommunications Software and Multimedia Laboratory at Helsinki University of Technology, PO Box 5400, FIN-02015 TKK, Finland.

Section 2 provides some background information regarding the authorization and access control mechanisms which have been widely used in centralized security schemes. It further reveals the characteristics that make them unsuitable for pervasive computing. Section 3 presents the system architecture of our distributed security scheme in terms of the key entities, their roles and their relations. A description of the prototype implementation of our distributed security scheme can be found in section 4. The paper concludes in section 5 with a short assessment of our security scheme and an evaluation of its future.

## 2 Background

Authorization and access control have been addressed in computer security from the early 60's [7]. However, the solutions that have been widely adopted in standalone systems and networked systems interconnected via, e.g., LANs and WANs fall short from dealing with the situations rising in pervasive computing. This section gives a short overview of these solutions and emphasizes their characteristics that make them unfit for pervasive computing.

### 2.1 Authorization

Security approaches to authorization can be broadly classified into two categories: identity-based and token-based. In this subsection we look at some representative examples from each category and we reveal the limitations of their application in pervasive computing.

#### 2.1.1 Identity-based approaches

Identity-based approaches rely on the authentication of the subject which is based on a user assigned identity (e.g. username) or on network addresses (e.g. virtual private networks or VPNs). Once the subject has been successfully authenticated, access control rules apply according to the subject's proven identity. Passwords have been widely used for authentication, but they require other means of authorization when they are issued to a user. In pervasive computing a subject may interact with a big number of peers. This is prone to result in the subject reusing one password for several peers, causing broad damages if the password is compromised. Using different passwords for each peer causes usability problems since the user must maintain a big number of passwords.

A widely used authentication mechanism in traditional networked system is Kerberos [8] where the authentication is based on usernames and passwords and the authorization to access services is based on ticket granting servers. In VPNs, IPsec [9] provides support for authentication in static environments, while enhanced mobileIP [10] can be used to support authentication in VPNs that include mobile devices.

Alternative authentication means are public key certificates, such as X.509 [11], which are based on a challenge-response protocol. However, X.509 requires a global wide hierarchical structure of Trusted Third Parties, which has not yet realized. Additionally, the names used in X.509 are in practice not globally unique.

#### 2.1.2 Token-based authorization

Token-based authorization divides authentication and authorization even further. Authorization is solely based on a token rather than on the identity of the bearer of the token. Popular means of token-based authorization rely on public key certificates. SPKI [12] is such an authentication scheme which is based on asymmetric key pairs and authorization certificates. The subject must prove that he possesses a certain asymmetric key pair, and present a SPKI certificate which binds a list of capabilities to that key pair.

The TeSSA security architecture [13] uses SPKI-certificates and policy management for authorization to provide a secure distributed computing environment. The Centaurus security architecture [5] is explicitly targeted for pervasive computing environments. Centaurus takes advantage of as well X.509-certificates as enhanced SPKI-certificates for authorization.

Another example of authorization based on certificates is the CBASS system [14] which demonstrates how certificates could be used in a real distributed system. There are also examples of authorization schemes that use other types of certificates. For instance, OASIS proposes in [15] appointment certificates which are used to authorize a subject for a certain role.

#### 2.1.3 The structure of an authorization scheme

The structure of an authorization scheme can be characterized in terms of distribution. In a centralized computing environment, which contains several terminals, there is an authorization manager, which the terminals can contact at any time. Whenever a user needs to access a service or an information, it contacts the central manager, which handles the authorization procedure and establishes the connection to the target service or information. Such proxy-based architectures have been proposed, e.g. [16]. These do, however, require the presence of a special proxy device, since the terminals in a pervasive computing environment, e.g. wearable devices, have too restricted computing power and storage space to be able to host a proxy. Yet, we cannot rely on the presence of such a terminal.

We solve this problem by introducing a separate authorization entity, which needs not be present when the information or services in fact are accessed.

## 2.2 Access Control

Access control is characterized by the granularity of the sets of subjects and objects to which it applies, the degree of discretion it allows to the object owner to provide access permissions and the intermediate levels it requires.

Given the ad hoc networking parameter in pervasive computing, the access control granularity must be fine-grained in order to allow greater flexibility in defining the access to services on a per-user basis. Employing attribute certificate schemes such as SPKI allows the application itself to define what fields are to be placed in the certificate. This in turn provides for a very fine-grained level of granularity suitable for pervasive computing.

Discretionary access control allows the owner of an object to define the access permissions on it. This leads to tailored security solutions for different classes of possible subjects that might require access permission to a given object [17]. However, that flexibility comes with a high implementation cost of system-wide discretionary policies for large systems [13]. Single, system-wide mandatory policies are less expensive to implement but they do not allow customized solutions for different subject classes.

To facilitate the access control process, intermediate levels such as roles or groups have been proposed (e.g. see [17, 15]). In these cases a subject is placed in a group or assigned a role and the access control is applied on groups and role and not on individual subjects. Combinations of both access control groups and roles provide for more flexible security solutions [13].

The more flexible, adaptable and customized access control is provided by a security solution, the more demanding and costly the implementation of the corresponding mechanism is. Considering the variety of cases where pervasive computing environments will be formed (e.g. home, office, travel, ad hoc contexts), the support for access control mechanisms must allow a variety of security policies to be applied. This would ensure that the security solution which provides the best trade-off between flexibility and cost in a given pervasive computing environment can be applied in it.

## 3 System Architecture

In this section we define a basic set of terms that will help us describe the system architecture of our authorization and access control scheme. Then we identify the entities that are involved in our security scheme and we delineate their roles and their relations.

### 3.1 System Model

The purpose of our system model is to provide the abstractions that describe the entities which participate in a pervasive computing environment. As a starting point, we adopt the various scenarios that are described by well-established

projects involved with pervasive computing (e.g. [2, 3, 4]). As our reference point we use the end-user who is interacting with a pervasive computing environment. In brief, these scenarios describe users which use personal, office and commercial information, while moving from one physical location to another. Hence, these scenarios can be abstractly described in terms of *users* and the *assets* (i.e. information) accessed by the users<sup>2</sup>.

Our system model provides two abstractions for capturing the above terms describing a pervasive computing environment. The *user* abstraction captures the entity which accesses (or attempts to access) some information. The *asset* abstraction captures all kinds of information that a user uses explicitly or implicitly, which can be personal or office data, advertisements, flight-schedule updates, etc. Both assets and users are addressable entities in a pervasive computing environment. Our system model also defines the *access* relation that captures the interaction between a user and an asset. A user accesses an asset when the entity represented by the user interacts (views, modifies, destroys, etc) the information held by the given asset. The access relation may take one the form of the following four actions:

- **create**: A user creates an asset when prior to this action the given asset did not exist, and the successful completion of the action resulted in the production of the given asset.
- **read**: A user reads an asset when he views the information held by the given asset.
- **write**: A user writes an asset he modifies the information held by the given asset.
- **delete**: A user deletes an asset when the given asset existed prior to the action, and the successful completion of the action resulted in the disappearance of the given asset.

Notice that these action are operations on assets, which are modeling vehicles that represent information. The create and the delete actions do not produce or destroy information; rather they produce or destroy the addressable carrier that may hold some information. It is the write action which places information in an already existing asset or destroys the information that an asset hold (e.g. by setting it to a void or otherwise useless value). From this perspective, it is through the read and write actions that the security properties of a piece of information can be compromised. For this reason, in the remainder of this paper we focus only on these two actions.

### 3.2 Roles and Relations

Unless all users are allowed to freely access in any way the information held by all assets, a security mechanism should

---

<sup>2</sup>N.B.: We are attempting to model a pervasive computing environment at a coarse-grained level; entities such as devices and services, as well as subtle notions of a context such as physical location, time, psychological condition of the end-user, etc are not addressed here.

be put in place to guarantee two fundamental security properties:

- **Confidentiality:** A piece of information can be disclosed only to *authorized* users.
- **Integrity:** A piece of information can be modified only by *authorized* users.

These definitions of the confidentiality and integrity properties reveal the three primary roles that are involved in our security scheme. The first role is that of the asset which holds a piece of information. The second role is the role of the user, who wants to read or modify the information held by an asset. There is also a third role, which is less explicit in the above definitions: The role of an authority. The authority is responsible to classify the information held by different assets, and to authorize different users to read or modify different information. In our system model the authority is a role played by a distinguished user. To summarize, an authority classifies assets and authorizes users to perform specific actions on specific assets. These roles and their relations are graphically illustrated in Figure 1 (a).

Notice, that the scheme allows for several authorities in a given environment. However, it is important to ensure, that for each asset there is a unique authority which will determine *which* users have the right to access it and *what* actions these users are allowed to perform on it. Therefore, a security policy is applied in each environment. The security policy determines the number of authorities in an environment and solves conflicts caused by authorities having conflicting asset access rules.

What makes the application of a security policy an interesting issue in pervasive computing is that the authority responsible for a specific asset may not be present when a user wants to access the given asset. In fact, we cannot even assume that these three entities (the user, the asset and the corresponding authority) could have directly interacted in the past. In our scheme, we ensure that the security policy in a given pervasive computing environment, in terms of access rights on assets, is respected, despite the fact that the security authority might not be present when the assets are accessed.

### 3.3 A Distributed Security Scheme

In our security scheme, assets are classified in groups, according to the set of users who have access on them, and the type of access (*read/write*) they have. If a user has access to one asset in a group, then that user has the same access to all assets in that group. Each group is assigned a pair of private encryption/decryption keys and all assets in that group are encrypted with the encryption key. A user is granted read access to a group of assets by obtaining the decryption key of the group. Accordingly, a user is granted write access to a group of assets by obtaining the encryption key of the group. Based on this scheme, different security policies can be applied.

A *flexible security policy* based on distributed trust can be developed on top of our distributed security scheme. A user can gain access to a certain asset by obtaining the corresponding key(s) from another user, with whom he has a mutual trust relation. This allows users to access assets without having to contact the security authority, a fact which makes the flexible security policy especially suited for pervasive computing.

A *conservative security policy* which imposes stricter authorization constraints than the above one, can also be implemented on top of our distributed security scheme. Before accessing assets in a pervasive computing environment, a user might have to present his credentials. These credentials have the form of a certificate, which the security authority has provided to users that it has successfully authenticated.

A *tight security policy*, which strengthens the security guarantees in each of the above cases, can also be implemented on top of our distributed security scheme. The keys associated with each access group can be time bound, allowing authorized users to access assets only within a specific time frame. This is especially useful for such pervasive computing environments where access to information is based on a membership, which must be periodically renewed.

In all three policies, the classification of the assets into different access groups and the issuing of the corresponding keys is done by the security authority. The security authority is also responsible for the initial distribution of these keys to users who are entitled to the corresponding authorization. Depending on the security policy, subsequent distributions of the keys may happen among trusted users. Finally, if a conservative security policy is used, the security authority also issues user certificates.

### 3.4 System Structure

The system that puts in place our distributed security scheme consists of three types of entities: the policy maker, the admission manager, and the participant.

The *policy maker* is a special user which plays a part of the security authority role. It is the one which defines the security policy that will apply to a given pervasive computing environment. This entity decides whether or not users must be authenticated before accessing assets, if key pairs are time bound, whether there is going to be a predefined number of access control groups of assets in a given environment or alternatively if each user is allowed to define access groups, etc.

The *admission manager* is also a special user in our system model, which plays the remaining part of the security authority role. This entity is responsible for generating the key pairs, and the user certificates if these are required. It is also responsible for authenticating users, and providing the users with the keys that the policy maker have entitled them to.

The *participant* is the user who wants to access assets in a pervasive computing environment. It may also possess assets that other participants in the same environment want to access. The participants obtain the appropriate keys, either from the admission manager or from other trusted participants, depending on the security policy. The participant is also responsible for the access control of the assets it possesses. Additionally, depending on the security policy, the participant may have to obtain a certificate for authentication purposes. This certificate can be obtained either from the admission manager or from another trusted participant, depending again on the applied security policy.

The way the above entities interact is graphically illustrated in Figure 1 (b). The policy maker contacts the admission manager to set the parameters that define the security policy that applies in a given pervasive computing environment. This may include the identities of the participants that are allowed to enter that environment and access assets in it, as well as the assets that each of these participants is allowed to access and the access rights on each asset.

Participants contact the admission manager to obtain the keys that enable them to access certain assets in a given environment. If required by the applied security policy, the participants have also to obtain their certificates for the given environment. Now, participants can contact other participants for obtaining the keys that would give them access to assets in the pervasive computing environment in question. Finally, participants contact other participants for accessing their assets.

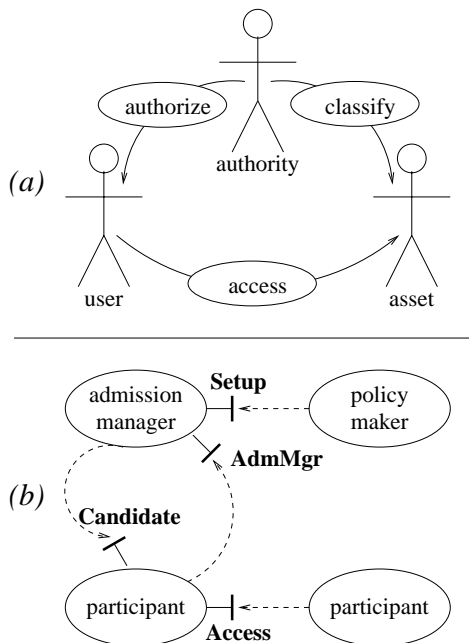


Figure 1. (a) The roles and their relations and (b) the system structure of the presented, distributed security scheme.

## 4 A Prototype

In this section, we describe how we implemented the entities described in the previous section. We first describe the interfaces between the entities, and move on to the implementation.

### 4.1 Interface Specifications

The definitions in OMG IDL syntax of all the interface described in this section can be found in Table 1.

#### 4.1.1 The Setup interface

The Setup interface is provided by the admission manager and used by the policy maker. It is used to setup the characteristics of the security policy that must be applied in a given pervasive computing environment. The policy maker configures the admission manager with an asymmetric key pair, the network address of the terminal the admission manager will run on, and an identifier which uniquely describes this pervasive computing environment. In addition, the policy maker defines the port number that participants can use to connect to other participants and to access their assets. Optionally, the policy maker can also specify a time limit until which the admission manager will accept requests of participants to access a given pervasive computing environment. This is especially useful in the case where the security policy allows participants to define their own access groups. In this case, the admission manager needs to know all the access groups before it generates and issues the corresponding keys.

#### 4.1.2 The AdmMgr interface

The AdmMgr interface is provided by the admission manager. It is used by participants who wish to obtain keys and possibly certificates in order to access assets in a given pervasive computer environment. The AdmMgr interface offers three services: authentication of itself (**authenticate**), registration of groups (**addGroup**), and enrolling the requests of participants (**enrol**) for keys and certificates. The **authenticate**-service uses a challenge-response protocol, and after successful authentication, the client receives a certificate for authentication in later communication. Additionally, in all service calls the participant must provide an identifier which describes the participant from whom the request originates. The admission manager can verify the identifier independently from the incoming request, and uses it in the authentication of the participant. This provides a protect against man-in-the-middle-attacks and against replay-attacks - assuming that the identifier resolution from the admission manager cannot be spoofed. This criteria cannot be fulfilled in many connections, i.e. in our implementation, the IP address of the participant serves the

purposes of the identifier. Also, in all service call the participant provides a signature which is a cryptographic signature computed over all parameters in the request. The signature secures the integrity and authenticity of the request but it does not protect against replay attacks.

### 4.1.3 The Candidate interface

The Candidate interface is provided by the participant. It is used by the admission manager mainly for authentication purposes. This interface offers three services: authentication of the participant (`authenticate`), acceptance of a certificate which is issued and offered by another party after it successfully has authenticated the client (`isAuthenticated`), and acceptance of an admission ticket which the admission manager grants after the enrollment phase has ended (`isRegistered`). For the same reasons as in the `AdmMgr` interface, in all service calls the admission manager provides its identifier and a signature.

The interaction between a participant and the admission manager takes place as follows: the participants starts by authenticating the admission manager through by calling `AdmMgr::authenticate`. The same call contains also the participant's public key and the access point where the admission manager can call back the participant. All parameters except for the signature are encrypted with the public key of the admission manager. The admission manager will respond with the challenge, encrypted with client's public key. In its turn, the admission manager will authenticate the participant in a similar way, calling the `Candidate::authenticate` service.

After the successful authentication of the participant, the admission manager creates a certificate if so compelled by the applied security policy. The certificate contains the identifier of the participant, the callback access point, the client's public key, and a signature. In addition, the certificate contains a UTC-time stamp whose purpose is to indicate a time after which the certificate is invalid. This field is used only when the applied security policy requires time bound certificates and access rights. The admission manager delivers the certificate to the participant by calling the `Candidate::isAuthenticated` service.

Once the participant has received the certificate it can call the `enroll` and `addGroup` services on the `AdmMgr` interface. When proposing an access group, the participant provides the public key for each other participant which shall have access to that group as well as the access rights of that participant. Group membership information is protected against disclosure by concatenating the public key and the access rights and encrypted together.

When the enrollment phase has ended, the admission manager creates admission tickets for all the successfully enrolled participants. A ticket contains a number of parameters related to the pervasive computing environment for which the ticket is issued (e.g. environment's identifier). Moreover, the ticket contains the keys that correspond to the access rights granted to the participant for each of the

access group that exist in the given environment.

### 4.1.4 The Access interface

The Access interface is provided by a participant to other participants. It is used every time a participant modifies or creates a new asset. That participant must then distribute the modified asset to all participants that are authorized to access that asset. Each asset has a two-part descriptor, which is unique to the given pervasive computing environment. That descriptor must be delivered along with the asset. For the same reasons as in the `AdmMgr` interface, the client must provide its identifier and a signature. The parameters are encrypted with the group encryption key and tagged with a group identifier before sending.

Participants, who do not have write access in a specific group, cannot encrypt assets with the encryption key of that group. Even if they can tag an asset with the correct identifier and send it to all participants, the participants cannot decrypt it with the correct key, and will therefore discard it. Accordingly, participants who do not have read access in a specific group, cannot decrypt assets accessible by that group, and can therefore not read them.

## 4.2 The implementation

Our prototype conforms to a conservative security policy. However, it is modular, and enhancing it to conform to several security policies is straight-forward.

In our prototype, the policy maker is a physical person that defines the security policy in a configuration file used to initialize the admission manager. On the other hand, the admission manager and the participant are software components which we have implemented as described in the remainder of this subsection. The relations between the units in the admission manager and the participant are depicted in Figure 2.

We use RSA encryption with PKCS-7 padding, and sign using SHA-1 with RSA. The same keys are used both for encryption and for signing. Whenever keys are passed between entities or stored in files, they are encoded using the standard RSA encoded key format.

### 4.2.1 The admission manager

The admission manager is a software component built with Java and consists of three parts: the server unit, the registry unit, and the policy checking unit. The server unit waits for requests from participants, verifies the signatures and converts the parameters to more abstract data representations, before passing on the data for processing by the session registry. The server unit also manages authentication issues and returns certificates to successfully authenticated participants.

The registry unit processes enrollment and group creation after checking with the policy checking unit the le-

<pre> <b>struct</b> AMParams {   <b>string</b> pubkey_fn;   <b>string</b> privkey_fn;   <b>string</b> net_address;   <b>long</b> session_id;   <b>long</b> session_port;   <b>long</b> reg_secs;   <b>string</b> ior_fn; };  <b>interface</b> Setup {   <b>void</b> begin( AMParams              params );   <b>void</b> interrupt(); }; </pre>	<pre> <b>typedef</b> <b>sequence</b> &lt;octet&gt;                 octets;  <b>interface</b> Access {   <b>void</b> receiveAsset (     <b>in</b> octets grpId,     <b>in</b> octets asset,     <b>in</b> octets desc_major,     <b>in</b> octets desc_minor,     <b>in</b> octets identifier,     <b>in</b> octets signature ); }; </pre>	<pre> <b>struct</b> Ticket { ... }; <b>struct</b> Cert { ... };  <b>interface</b> Candidate {   octets authenticate (     <b>in</b> octets chall,     <b>in</b> octets identifier,     <b>in</b> octets signature );   <b>void</b> isAuthenticated (     <b>in</b> Cert cert,     <b>in</b> octets identifier,     <b>in</b> octets signature );   <b>void</b> isRegistered (     <b>in</b> Ticket ticket,     <b>in</b> octets identifier,     <b>in</b> octets signature ); }; </pre>	<pre> <b>interface</b> AdmMgr {   octets authenticate (     <b>in</b> octets session_id,     <b>in</b> octets challenge,     <b>in</b> octets key,     <b>in</b> octets access_pt,     <b>in</b> octets id,     <b>in</b> octets signature );   <b>void</b> addGroup (     <b>in</b> MemberList list,     <b>in</b> Cert cert,     <b>in</b> octets id,     <b>in</b> octets signature );   <b>void</b> enroll (     <b>in</b> Certificate cert,     <b>in</b> octets identifier,     <b>in</b> octets signature ); }; </pre>
<b>Setup</b> interface	<b>Access</b> interface	<b>Candidate</b> interface	<b>AdmMgr</b> interface

Table 1. The interfaces defined in our distributed security scheme.

gitimacy of such requests. Legitimate requests are honored and the registry unit keeps information about the enrolled participants and the created groups. It is also the responsibility of the registry unit to use this information in order to create the appropriate admission tickets for enrolled participants. This is again done in cooperation with the policy checking unit. The policy checking unit is the part of the admission manager that get configured by the input provided by the policy maker (i.e. the configuration file of the system in our implementation).

#### 4.2.2 The participant

The participant is a software component implemented using C++ and consists of three parts: the communication management unit, the asset registry unit, and the proxy unit. The communication management unit is responsible for processing incoming and outgoing assets. Incoming assets are decrypted with the appropriate group decryption key, and passed on to the asset registry unit. Outgoing asset are encrypted using the proper group encryption key, and then send to their destination participant.

The asset registry is responsible for creating a unique descriptor for the assets that the proxy unit creates. These asset identifiers consists of two parts: the client's unique identifier and a client specific sequence number. The asset registry unit also keeps track of the received assets and the access group to which they belong. If at a later point and asset is received with the same identifier but with different access group identifier, the asset registry unit discards it as illegitimate. The proxy unit is responsible for passing assets between the registry unit and the user.

## 5 Conclusion

This paper presented an enhanced authorization and access control scheme for pervasive computing, which uses a centralized security authority only off-line and not when assets are accessed. The authority can apply different security policies with authorization and access control schemes that fit different environments. The access control process is completely distributed, based on the encryption of assets and the distribution of encryption/decryption keys to the participants with adequate access rights.

Our security scheme supports time-bound encryption keys and provides to the access control mechanism the possibility to require a user certificate prior to the distribution of the encrypted asset. These choices provide for the employment of a variety of security policies, each with a different degree of constraints to the access of assets and a different level of security guarantees for them.

Our prototype implements this scheme elegantly, although certain enhancements are still needed. For example the encryption mechanism must be tuned so as to be fast enough for limited-resource devices. It must also be enhanced in order to produce non-identical output for the same input, as this property weakens confidentiality.

To validate the proposed security scheme we are employing it in the pervasive computing environment that is formed for a business-meeting scenario. The policy maker in this scenario is the person who call the meeting and who sets the time and the place for it. Both cases of invited participation as well as spontaneous participation are considered. Each invited participant is allowed to form his own access groups and spontaneous participant may obtain the necessary keys from the group owners. The target platform

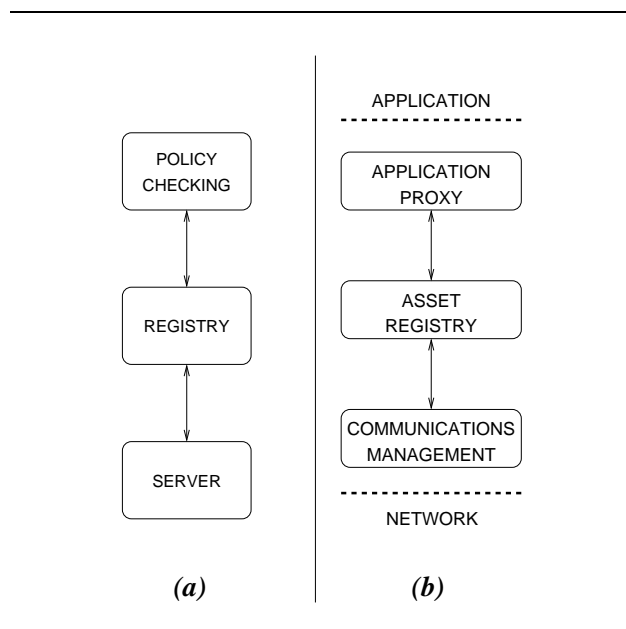


Figure 2. The units in the admission manager component (a), and the participant component (b).

for the participants is a configuration of Linux-based Compaq iPaqs communicating over Bluetooth, while for the admission manager the target platform is the Java Runtime Environment on a regular workstation.

## Acknowledgements

The authors thank Jouni Karvo, Sanna Liimatainen and Teemupekka Virtanen for their valuable comments.

## References

- [1] M. Satyanarayanan. Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.
- [2] Massachusetts Institute of Technology. The MIT Project Oxygen. <http://oxygen.lcs.mit.edu/>.
- [3] Carnegie Mellon University. Project Aura: Distraction-free Ubiquitous Computing. <http://www-2.cs.cmu.edu/aura/>.
- [4] Berkeley University of California. The Endeavour Expedition: Charting the Fluid Information Utility. <http://endeavour.cs.berkeley.edu/>.
- [5] L. Kagal, T. Finin, and A. Joshi. Trust-Based Security in Pervasive Computing Environments. *IEEE Computer*, 34(12):154–157, December 2001.
- [6] T. Saridakis. Nomadic Collaboration Management. In *Proceedings of the 3rd International System Administration and Networking Conference (SANE 2002)*, pages 259–273, May 2002.
- [7] D. Gollmann. *Computer Security*. John Wiley & Sons, 1999.
- [8] J. Kohl and C. Neuman. The Kerberos Network Authentication Service V5 (RFC 1510). Technical report, Network Working Group, September 1993. <http://www.ietf.org/rfc/rfc1510.txt>.
- [9] R. Thayer, N. Doraswamy, and R. Glenn. IP Security Document Roadmap (RFC 2411). Technical report, Network Working Group, November 1998. <http://www.ietf.org/rfc/rfc2411.txt>.
- [10] V. Gupta and G. Montenegro. Secure and Mobile Networking. *ACM Mobile Networks and Applications*, 3(4):381–390, 1999.
- [11] A. Arsenalut & S. Turner. Internet X.509 Public Key Infrastructure: Roadmap. Technical report, PKIX Working Group, July 2002. <http://www.ietf.org/internet-drafts/draft-ietf-pkix-roadmap-09.txt>.
- [12] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory (RFC 2693). Technical report, Network Working Group, September 1999. <http://www.ietf.org/rfc/rfc2693.txt>.
- [13] P. Nikander. *An Architecture for Authorization and Delegation in Distributed Object-Oriented Agent Systems*. PhD thesis, Helsinki University of Technology, Department of Computer Science, Telecommunications Software and Multimedia Laboratory, 1999. <http://www.tml.hut.fi/pnr/publications/PhDThesis.pdf>.
- [14] J. Dai and J. Alves-Foss. Certificate Based Authorization Simulation System. In *Proceedings of the 25th Annual International Computer Software and Applications Conference (COMPSAC 2001)*, pages 190–195, October 2001.
- [15] J. Bacon, K. Moody, and W. Yao. A Model of OASIS Role-Based Access Control and Its Support for Active Security. *ACM Transactions on Information and System Security*, 5(4):492–540, November 2002.
- [16] M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, and R. Rivest. Proxy-Based Security Protocols in Networked Mobile Devices. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 265–272, 2002.
- [17] P. Fenkam, S. Dustdar, E. Kirda, G. Reif, and H. Gall. Towards an Access Control System for Mobile Peer-To-Peer Collaborative Environments. In *Proceedings of 11th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2002*, pages 95–100, June 2002.