Authorization Management for Pervasive Computing

Patroklos Argyroudis

A thesis submitted to the University of Dublin, Trinity College

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

October 2006

Declaration

I, the undersigned, declare that this work has not previously been submitted to this or any other University, and that unless otherwise stated, it is entirely my own work.

Patroklos Argyroudis

Dated: 19th October 2006

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Patroklos Argyroudis

Dated: 19th October 2006

Acknowledgements

The are certain people that I would like to thank for their support during the course of my postgraduate studies and the completion of this dissertation. First of all, I would like to thank my supervisor, Prof Donal O'Mahony. His advice and guidance have been invaluable, not only in research matters but also in my general academic life. He helped me to concisely formulate my ideas and identify their strong and weak points, even when they were outrageous. I can only hope to some day be half the supervisor he has been to me.

One of the most influential and supportive people during my studies has been Ioanna Stamouli. Her constant moral support, especially during all those times that self-doubt almost took over, gave me courage and kept me going.

Dr Linda Doyle has also been very helpful throughout my research studies. Her insightful comments led me to appreciate areas not directly related to my work but nonetheless very important. I would like to thank her for this.

I have been very lucky to conduct my research among the extremely talented people of the Networks and Telecommunications Research Group (NTRG). Hitesh Tewari, Diego Doval, Stephen Toner and Raja Verma, among others, were always happy to discuss technical topics, ideas and help me enhance my understanding of areas outside my main research focus.

During my time in Dublin several of my friends from Greece were always willing to help me in various ways. I would like to thank them all. In particular, Georgios Samaras has been very helpful with the BNF notation of the ÆTHER policy language. Also, Dimitris Glynos, Kiriakos Oikonomakos and Georgios Kargiotakis somehow always managed to find the time to discuss technical problems.

I wish to thank the Embark Initiative, operated by the Irish Research Council for Science, Engineering and Technology (IRCSET), for providing financial support for my studies.

Finally, I would like to dedicate this work to my family. My parents, Georgios and Stavroula, and my sister, Thomi-Niki, supported me in countless ways. Their unconditional love helped me to make it through all the tough times in my life. I wholeheartedly thank them.

Patroklos Argyroudis

University of Dublin, Trinity College

October 2006

Contents

List of Figures

List of Tables

Abstract

The technological developments of the last decade, particularly in the field of mobile microprocessor design, have enabled the integration of information processing capabilities in small everyday devices and appliances. This trend is leading the computing world to a paradigm shift; the computer is no longer a personal dedicated device used for specific operations, but it is woven into the surrounding environment providing its services in a transparent manner. The pervasive computing paradigm approaches computing from a human-centric, non-intrusive viewpoint. However, its extremely dynamic and open nature raises security and privacy concerns that need to be addressed in a coherent manner along with the development of the required infrastructure. Although the traditional security requirements remain the same, this new approach to computing has introduced additional challenges.

The main problem in addressing the security requirements of pervasive environments is the large number of ad hoc interactions among previously unknown entities, hindering the reliance on predefined trust relationships. As users roam among administrative domains the devices they carry must be able to interact in a disconnected and decentralized manner. Secure connections must be established spontaneously, without the need to access online central entities. The problem is aggravated since data transmissions use wireless media, such as Bluetooth and IEEE 802.11, whose integrity and confidentiality can easily be undermined by malicious entities. The proposed security solutions must also take into account the ubiquitous computing vision that demands interaction interfaces that integrate naturally with the goals users are trying to achieve in order to be as unobtrusive as possible. Perceived contextual information from the environment should be utilized in order to facilitate simplified management and to minimize human intervention for the

required administrative tasks. Existing security management approaches fail to capture the complete set of these requirements.

In this dissertation we propose *ÆTHER*, a novel authorization management framework designed specifically to address trust establishment and access control in pervasive computing environments. ÆTHER directly extends the traditional RBAC model to support decentralized administration, disconnected operation and context-awareness. Furthermore, we use the well-defined concept of location-limited channels to specify an unobtrusive usage model for the required administrative tasks. Based on this general framework we have instantiated two different systems. The first one, ÆTHER_0, has been designed to address the authorization requirements of small pervasive environments which consist of particularly constrained devices. The second, ÆTHER_1, addresses the authorization requirements of large pervasive computing domains that have multiple owners with complicated security relationships. Our implementation and evaluation of the two instantiations demonstrates the feasibility of deploying and using them in real-world pervasive computing environments.

Chapter 1

Introduction

The pervasive computing paradigm predicts that in the near future we are going to be surrounded by countless wireless devices capable of providing services transparently. Computing devices are being embedded into everyday appliances and become part of our environment. Interactions with such devices must be integrated with the purpose a user aims to achieve in a natural, graceful way in order to *feel* ubiquitous. However, the open nature of such environments raises security concerns that need to be addressed in a coherent manner along with the development of the required underlying infrastructure. Although the traditional security requirements remain the same, this new approach to computing has introduced additional challenges.

The main problem in addressing the security requirements of pervasive computing environments is the large number of ad hoc interactions among previously unknown entities[1], hindering the reliance on predefined associations. Authentication mechanisms can be employed to establish the identity of a pervasive computing entity but they suffer from scalability problems and have limited value in defining authorization decisions among strangers. Another equally important problem is that the employed security solution should follow the ubiquitous computing vision and be naturally integrated with the actions the users perform in order to complete their objectives. A user that carries a multitude of devices must be able to establish spontaneous secure communication channels with the devices embedded into the environment or carried by other users without extensive manual recon-

[1] That is entities that visit the current domain for the first time or entities that have visited the domain in the past but no state information about them has been saved.

figuration tasks. Contextual information perceived by the devices from the environment should be employed in order to enable such communication needs.

According to our vision the global pervasive computing infrastructure will be built in a non-hierarchical manner by connecting different autonomous domains via the existing mobile and fixed communication networks. The security considerations associated with this vision must be addressed in parallel with the development of the architecture designs that are going to support its employment. Consider the following simple but typical scenario in a pervasive computing environment (illustrated in Fig. 1.1). The owner wishes to control a light switch through her wristwatch via a wireless transmission medium and also wants to allow a trusted visitor in the house to do the same. However, in both cases she wants to be sure that a malicious party who walks outside the house will not be able to control the light switch and capture or modify any of the above wireless transmissions. Additionally, the necessary administrative actions should be kept to an absolute minimum.

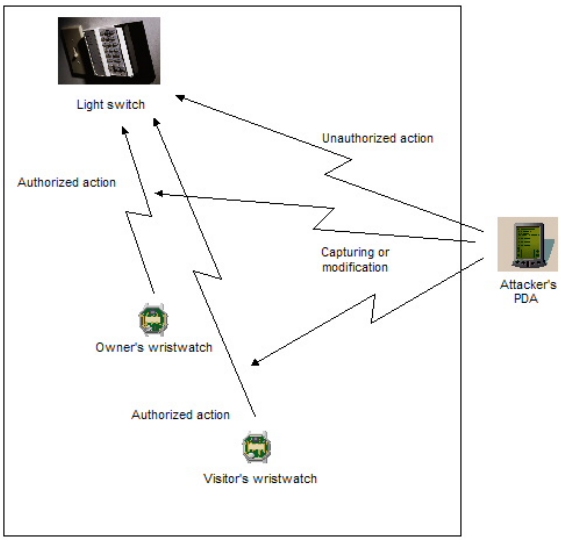

Figure 1.1: Typical insecure pervasive computing scenario.

In this dissertation we propose ÆTHER[2], an authorization management framework designed specifically to address trust establishment and access control in pervasive computing environments where *a priori* knowledge of the complete set of participating entities

[2]The name was inspired by the medium that was once believed to pervade all space supporting the propagation of electromagnetic waves.

and global centralized trust registers cannot be assumed. The basis of our work is the Role-Based Access Control (RBAC) model [FK92], according to which entities are assigned to roles and roles are associated with permissions. ÆTHER extends RBAC in order to support decentralized administration, disconnected operation and context-awareness. Furthermore, we use the well-defined concept of location-limited channels to specify an unobtrusive usage model for the required administrative tasks.

Based on this general framework we have instantiated two different systems:

1. ÆTHER$_0$ has been designed to address the authorization needs of small pervasive environments whose management requirements are simple. It utilizes only symmetric key cryptography in order to provide security services. Consequently, it is appropriate for devices that have particularly limited processing capabilities (such as simple sensors).

2. ÆTHER$_1$ addresses the authorization requirements of large pervasive computing domains that have multiple owners with complicated security relationships. It relies on asymmetric cryptography and therefore is more fitting to domains that consist of devices that have sufficient information processing capabilities.

1.1 Requirements

In order to specify and build an authorization management framework that is able to support pervasive computing, a number of requirements must be taken into consideration:

- Decentralized management. The dynamic nature and the great number of participating devices in pervasive computing imply the need for frequent establishments of communication channels between entities that belong to different administrative domains. Centralized architectures are not able to support the autonomous non-hierarchical building of a secure pervasive computing infrastructure. Furthermore, external centralized entities constitute a single point of attack/failure and it is not realistic to assume that they can be universally trusted.

- Disconnected operation. This requirement is closely related to the previous one. An authorization architecture for pervasive computing must be able to support the

establishment of secure relationships between devices that are not able, due to physical location, network partitions, or other reasons, to connect to a third party for mutually authenticating and authorizing each other.

- Unobtrusiveness. Human intervention for the required administrative tasks should be naturally and gracefully integrated with the physical workflow of users in order to facilitate the unobtrusive nature of pervasive computing.

- Context-awareness. The use of contextual information such as physical location, activity and others, should be utilized to allow the dynamic adaptation of the management framework to a variety of situations and applications. Moreover, context-awareness minimizes human intervention and user distractions for configuration purposes alleviating unobtrusiveness.

1.2 Summary of Goals

Our goals for the work presented in this dissertation are summarized below:

- The identification and analysis of the requirements of authorization management in pervasive computing. Furthermore, the examination of existing security models and systems with regard to the previously identified requirements.

- The definition of a pervasive computing threat model that will allow us to identify the specific attacks and avenues of attack that we will try to offer protection against.

- The development of an authorization management framework for pervasive computing based on the identified requirements and the formulated threat model.

- The demonstration of the developed system's feasibility through a real-world implementation and evaluation.

1.3 Key Contributions

The key contribution of this thesis is the presentation of the design and engineering of a complete authorization management system for pervasive computing environments. We

can identify our significant contributions within this scope in the following five areas:

1. The theoretical ÆTHER framework that extends the traditional RBAC model to include support for decentralized management, disconnected operation, context-awareness and unobtrusive administration.

2. Our extensions to the concept of location-limited channels to use them as part of users' physical actions to state their intent regarding authorization while being integrated gracefully with the primary task, thus achieving unobtrusive security management.

3. Our enhancements to the KeyNote trust management system to allow its use in realizing both the traditional RBAC authorization model and our extensions to it. Specifically, we have modified KeyNote's policy language, and extended its inference engine to include support for dynamic attribute authorization decisions, dynamic values for the utilized attributes, and integer delegation control.

4. The ÆTHER_0 management model that relies exclusively on symmetric cryptography to instantiate our general framework.

5. The ÆTHER_1 instantiation and its novel RBAC delegation model based on the concept of dynamic Authority Attribute Sets (AASs).

1.4 Dissertation Outline

The rest of this dissertation is structured as follows: This chapter briefly introduces the pervasive computing paradigm and the associated security challenges. It also summarizes the requirements for a pervasive computing authorization management architecture and presents our goals for the rest of the work.

Chapter 2 sets the scene of pervasive computing and the related enabling concepts of context-awareness and service discovery. It also examines several existing pervasive computing systems in order to clearly define the target area. The main focus of the chapter is the survey of traditional and pervasive computing security systems that have

been published in the literature. The related open research problems are identified by presenting a thorough classification of the examined projects.

Chapter 3 presents a comprehensive threat model for pervasive computing. A threat model is required in order to clearly define attack avenues, specific attacks and the related assumptions. This analysis helps us to gain a better understanding of a pervasive computing environment from the attacker's viewpoint and directly leads to the formulation of a security model.

Chapter 4 introduces a novel authorization management framework that has been designed to address the requirements of pervasive computing environments. The general framework is then instantiated into two distinct systems that address the security and the user demands of different pervasive authority domains. Moreover, usability and maintainability aspects are analyzed focusing on the needs of the end users. The chapter ends by offering a detailed comparison of the two instantiations and an examination of which one is applicable to which pervasive computing scenario.

Chapter 5 presents the implementation of two prototypes for the two instantiations of the ÆTHER framework and examines their feasibility using modern handheld devices as the development hardware platform. Furthermore, the chapter demonstrates that both implemented solutions perform more than adequately well without introducing significant computation overhead or user distractions.

The final chapter discusses the conclusions of the work presented in the dissertation and summarizes the contributions. Finally, it identifies areas that future research on the subject should focus on.

1.5 Publication Record

Towards the completion of the work presented in this dissertation we have published subsets of the produced research in refereed international journals, conferences and workshops:

- Patroklos Argyroudis and Donal O'Mahony, "Secure Routing for Mobile Ad hoc Networks", *IEEE Communications Surveys and Tutorials*, IEEE Press, vol. 7, no. 3, pp 2-21, 2005.

- Patroklos Argyroudis and Donal O'Mahony, "Towards Flexible Authorization Management", In *Proceedings of 10th IEEE International Symposium on Computers and Communications (ISCC'05)*, IEEE Press, pp 421-426, 2005.

- Ioanna Stamouli, Patroklos Argyroudis and Hitesh Tewari, "Real-time Intrusion Detection for Ad hoc Networks", In *Proceedings of 6th IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WOWMOM'05)*, IEEE Press, pp 374-380, 2005.

- Patroklos Argyroudis and Donal O'Mahony, "Towards a Context-aware Framework for Pervasive Computing Authorization Management", *3rd UK-UbiNet Workshop: Designing, Evaluating and using Ubiquitous Computing Systems*, 2004.

- Jean-Marc Seigneur, Patroklos Argyroudis, David O'Callaghan and Joerg Abendroth, "The REL Project: Mobile-based Reliable Relations", *1st Workshop on Friend of a Friend, Social Networking and the Semantic Web*, 2004.

- Patroklos Argyroudis, Raja Verma, Hitesh Tewari and Donal O'Mahony, "Performance Analysis of Cryptographic Protocols on Handheld Devices", In *Proceedings of 3rd IEEE International Symposium on Network Computing and Applications (NCA'04)*, IEEE Press, pp 169-174, 2004.

- Patroklos Argyroudis and Donal O'Mahony, "Securing Communications in the Smart Home", In *Proceedings of 2004 International Conference on Embedded and Ubiquitous Computing (EUC'04)*, LNCS 3207, Springer-Verlag, pp 891-902, 2004.

- Patroklos Argyroudis and Donal O'Mahony, "ÆTHER: an Authorization Management Architecture for Ubiquitous Computing", In *Proceedings of 1st European PKI Workshop: Research and Applications (EuroPKI'04)*, LNCS 3093, Springer-Verlag, pp 246-259, 2004.

We have also contributed to the following Computer Science Department Technical Report:

- Jean-Marc Seigneur, Anselm Lambert, Patroklos Argyroudis and Christian D. Jensen, "PR3 Email Honeypot", Technical Report TCD-CS-2003-39, Department of Computer Science, University of Dublin, Trinity College, 2003.

Chapter 2

Background and Related Work

The main goal of this chapter is to survey the existing research work in the area of pervasive computing authorization systems, identify their strengths and shortcomings, and consider whether they satisfy the requirements of the human-centered computing paradigm. Consequently, we first analyze the properties of pervasive computing in general and the related enabling technologies, such as context-awareness and service discovery. To gain a more clear understanding of our target area, we also examine several existing ubiquitous computing systems and identify possible security threats associated with the functionalities they offer. We then offer a brief overview of computer security and its conventional goals that are relevant to our research area, as well as traditional computer security systems and the reasons they fail to address pervasive computing requirements. The main focus of the chapter is the survey of pervasive computing security systems that have been published in the literature. The chapter ends with a thorough classification of the examined projects and an identification of the related open research problems.

2.1 Pervasive Computing

The technological developments of the last decade, particularly in the field of mobile microprocessor design and wireless networking, have enabled the integration of information processing capabilities in small everyday devices and appliances. This trend is leading the computing world to a paradigm shift; the computer is no longer a dedicated device used for

generic operations, but it is woven into the surrounding environment providing its services in a transparent and unobtrusive manner. The ubiquitous, or pervasive (as it has been known since the mid-1990s) computing paradigm envisioned by Mark Weiser [Wei91] approaches computing from a human-centric, non-intrusive viewpoint: "The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it." According to Weiser's vision, future computing environments would be composed of cheap and disposable devices interconnected via wireless media. These devices would either be embedded into the environment itself or worn by people. As it is clearly obvious, this part of Weiser's vision is already a reality. Embedded computing has indeed become pervasive. In our homes we have countless devices that use microprocessors for management and controlling functions. Furthermore, the number of information processing devices that we carry with us is also increasing. Sophisticated mobile phones, MP3 players and PDAs are quickly becoming a necessity for supporting modern human activities. Even the interconnection requirement is a reality. Wireless technologies such as IEEE 802.11 and Bluetooth enable the interconnection of our mobile phones with headphones and our PDAs with nearby printers.

However, the main characteristic of Weiser's vision that concerns the usage model of ubiquitous computing systems has yet to be engineered. Future computing environments will not be comprised of traditional dedicated computers, but rather of everyday devices and appliances with information processing and communication capabilities. Users will operate these devices according to usage models that integrate naturally with the actions they perform in order to complete their objectives. Thus, computing will become ubiquitous as it will support the activities of the users without being a distraction. The design of such computing usage models, often called *calm* computing [WB96], will allow a user to change her focus from the system to another activity in a fluid and natural way. According to previous research efforts [PRM99], [SAW94], [KF02], in order to allow ubiquitous computing systems to be unobtrusive to the user and adaptive to his ongoing activities the ability to sense, collect and process context is fundamental.

Based on the above discussion we now enumerate the following three key properties of pervasive computing systems:

1. Extremely open and dynamic nature. Pervasive computing environments are comprised of countless devices that are able to process information, interconnect with others, offer and consume services. These devices are not bound to a specific physical area; people carry them around as they roam among administrative domains. This suggests large number of interactions among unknown entities, that is entities that do not belong to the same administrative domain and have not communicated before, or have done so but the state of their relationship has not been maintained. Two related important requirements of computing systems designed for open and dynamic environments are disconnection and decentralization. Connectivity between devices in pervasive computing must be established without the need to contact online central servers for configuration, security or other reasons.

2. Graceful and unobtrusive usage models. The interaction interfaces of the devices specifically, and more generally the usage models of pervasive computing systems, should be integrated naturally with the goals the users are trying to achieve in order to minimize human distraction and make ubiquitous computing *feel* ubiquitous.

3. Context-awareness. Pervasive computing systems and applications should be able to discover and utilize contextual information (like user location, current activity, and time of day). This will allow them to dynamically adapt to usage scenarios and support the users' information processing requirements without becoming a distraction.

These properties in essence lead to human-centered computing. Instead of having the user interact with a dedicated machine and focus all his attention on it, handheld or embedded devices will perform the user's requests, and more importantly the user's *will*, without requiring constant concentration and extensive manual tasks.

Although a lot of researchers include hardware constraints, such as limited processing and memory resources, in the list of pervasive computing inherent properties, the constant advancement of hardware technology increasingly makes this an irrelevant issue. As an example consider that in 2005 the available high-end devices for smartcard products were using up to 100 MHz CPU clock speeds, while a year before the typical top speed for the same class of devices was 33 MHz [Mit04]. Furthermore, both static and volatile memory

is constantly increasing enabling pervasive computing programmers and system designers to worry less and less about cache sizes and storage requirements.

Another critical goal, but not necessarily a key property, of pervasive computing is *programmability*. Integrated smart spaces should allow application programmers to perceive them as space-specific software libraries to be used from a high-level language [Hel05]. The devices, sensors and actuators of the smart space should be represented as services that can easily be accessed and used by developers. In this *service-oriented* view of pervasive computing every available information gathering or processing resource could be composed with every other, with which is compatible, enabling rapid prototype development. Thus, a pervasive computing developer could create applications that for example adjust the light level of the room when the TV is on by composing the access interfaces of the room-light sensor service, a window-blind sensor service and the TV [Hel05]. Service discovery frameworks can facilitate the standardization of access interfaces as well as of the protocols for the location of these services in dynamic environments.

2.2 Context and Context-awareness

As we have seen in the previous section, the ability to discover and adapt to contextual information is one of the key properties of ubiquitous computing systems. For humans, context recognition and adaptation is a natural process that happens frequently during daily activities. When we want to divulge a secret to a trusted colleague we are careful to note the people in our vicinity and lower our voice. However, in computer science the term "context" is usually vague and overloaded with many different meanings. For the purpose of this thesis we will ignore context as defined in other areas of computer science, and focus only on context that is or can be utilized by ubiquitous computing systems.

There have been many attempts in the literature to give a formal definition of context. Instead of trying to include them all here, we will focus on the most representative ones. In [DA99] context is defined as:

> "Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the inter-

action between a user and an application, including the user and applications themselves."

Chen and Kotz give the following definition [CK00]:

> "Context is the set of environmental states and settings that either determines an application's behavior or in which an application event occurs and is interesting to the user."

As it is obvious from the given definitions, context is difficult to be defined generally since it is highly relevant to the situation and system use case under investigation. In order to give a definition of context that is more appropriate and useful for systems development, Adelstein et al. [AGRS05] based on work by Schilit [SAW94] and Chen [CK00], divide context into two categories. The *enumeration-based* category defines context by employing five categories, as follows:

1. Computing context, which includes information regarding network connectivity, communication costs, available bandwidth, and nearby accessible resources such as printers, file servers and others.

2. User context, such as profiles with user defined settings, current location, and other users within the vicinity of the user.

3. Physical context includes level of lighting and noise at the current location, temperature, etc.

4. Temporal context, such as time of day, week, month, season of the year.

5. Context history, which are the data collected across a time span according to the above four context categories.

The *role-based* category for defining context does so by explicitly labelling context data according to how it can be used by a pervasive computing application. Consequently, *active context* is the contextual information used by an application to adapt its behavior, and *passive context* is the contextual information that although it is relevant to the user for

enhancing her understanding of the environment, it is not used for application adaptation [AGRS05]. The enumeration and role based categories are clearly not mutually exclusive; Room temperature, being a physical context, can be used by an application as active context in order to adapt the settings of an air conditioner.

Another important aspect of context is its ability to be aggregated in order to form more complete representations and understanding of the current user environment. Low-level contexts provided directly by sensors, such as location, time, temperature, and others, can be aggregated into other higher-level sources of contextual data. For example, by aggregating a user's current time and location, along with her calendar, a pervasive computing application can derive the user's current activity, like attending a meeting, or sitting in class [CK00]. Service-orientation, as explained in the previous section, can facilitate context aggregation by composing the state of several sensors available in a smart space.

Context definitions are useful for knowing what types and categories of contexts are available to pervasive computing applications. However, the problem of effectively utilizing these to respond to changes in the user's environment to enhance her understanding or to minimize configuration tasks still presents challenging research questions. Context-awareness includes the following four capabilities according to Pascoe [Pas98]:

1. Contextual sensing, which is the capability of detecting environmental states and settings. This also includes the way that this information is presented to the user.

2. Contextual adaptation, the capability of an application or system to dynamically adapt its behavior according to sensed context.

3. Contextual resource discovery, refers to the discovery of available and accessible resources that can be used by the application in order to facilitate better adaptation to the user's needs.

4. Contextual augmentation, which is the capability of associating digital data with a user's current context.

Another definition of context-aware computing is given by Chen and Kotz in [CK00]. By using the role-based context definition they identify two ways of using context: dynamic

adaptation of the application's behavior according to sensed context data, and presentation of the context data to the user (or storing for later retrieval). Thus, context-awareness can be divided into the following two categories:

1. Active context-awareness, according to which an application dynamically adapts to sensed context and changes its behavior.

2. Passive context-awareness, in which an application stores or presents sensed context information to the user.

In pervasive computing active context-awareness can be used to develop adaptive applications. The optimum goal is to minimize the need of extensive manual configuration tasks and naturally integrate the ones that are required to the physical world actions the users perform to achieve their objectives. In our developed architecture we only focus on active context-awareness, as it can be directly used to develop adaptive security policies and minimize the need for reconfiguration.

2.3 Service Discovery

As the vision of pervasive computing becomes a reality the number of devices in our homes and in the environments we visit constantly increases. Since the goal of pervasive computing is to make devices and information processing capabilities disappear, these devices need to utilize wireless media to offer their services in a transparent manner. In the past, workstation computers were physically connected via wires to peripheral devices such as printers and scanners. Although wired connections provide ready access to the services of the connected devices, they limit mobility and prevent natural integration with the surroundings. Moreover, human intervention is frequently needed in order to maintain these devices and enable them to interoperate with each other and with the central workstation computer. In a computing environment where there are hundreds of participating devices these limitations present serious problems. Users cannot be realistically expected to constantly maintain and configure all of their devices. Mobile devices entering and leaving the environment have to be able to dynamically interact with it and discover other devices and provided services as needed.

As a result, service discovery is an enabling technology for pervasive computing. For example, the lights of a room may be automatically turned off as the user leaves the room and storage space on a file server can be requested when a PDA cannot accommodate its user's needs. Service discovery frameworks, such as Jini [ASW+99], Universal Plug and Play (UPnP) [UPn03], and the Service Location Protocol (SLP) [GPVD99], aim to standardize software interfaces and protocols for service discovery creating interoperability and making their implementation straightforward. When a device has support for service discovery it usually means that it is able to support a subset of the following capabilities:

- Standardization of service definitions. In order for a provided service to be discovered and used dynamically it must be defined according to a universally agreed way. A service definition includes attributes or textual messages that describe what exactly is provided, the access interface and the operations it supports, the utilized protocols, as well as other related information.

- Search and location protocols. A service discovery framework must provide the ability to locate required services on demand. This is usually done by allowing users to specify in textual descriptions what are they looking for. User input may be completely free-form, such as "color printer", or structured in attribute name-value pairs, for example "device:printer, capability:color".

- Advertisement of services. The capability of service advertising allows the dynamic update of the local networking environment with insertion and deletion of available services. When a device is turned on, or comes in the vicinity of a network, it advertises the services it provides according to its policy configuration. The generated advertisements contain service availability data that are either delivered directly to other client devices, or to dedicated devices that register and maintain lists of available services. We must note here that service advertisement and discovery is characterized by its dynamic nature. Traditional information dissemination systems, such as the Domain Name System (DNS) [Moc87] and the Dynamic Host Configuration Protocol (DHCP) [Dro97], are based on static configuration files that have to be maintained by human administrators. Service advertisement and discovery

in pervasive computing implement a more dynamic model according to which available information may be inserted or deleted from the network with no or minimal intervention by regular users without special authority over the system.

- Service registration. As mentioned above, when a service advertisement is generated it can either be broadcasted into the network where it can reach possible service consumers directly, or it can be sent to a dedicated preconfigured device that catalogs all available services. In the latter case, clients locate required services by querying the catalogs instead of broadcasting queries in the network. Although service catalogs require preconfigured infrastructure, they offer certain advantages to a pure ad hoc service discovery approach. For example, the overhead of broadcast traffic in the network is significantly reduced. Also, the synchronization of service availability among different networks is greatly simplified by just processing the catalog of the dedicated device responsible for each network. However, the need for configuration and maintenance of the service cataloging devices introduces an additional administration burden to a user. Therefore in pervasive computing environments where the minimization of human intervention is fundamental, ad hoc approaches to service advertisement and location are preferred.

Since service discovery in pervasive computing is inherently dynamic and free of globally accessible registries that are always online and reachable, the provided services must adhere to certain properties. These properties are used to enable the identification, location and standardization of services in such volatile environments.

Identification of the provided services is essential in service discovery frameworks. In large pervasive computing environments that span across many physical spaces it is possible to have the same service provided by two different devices. One method to distinguish between services is to assign to them Universally Unique Identifiers (UUIDs). The use of UUIDs allows devices to search for specific services without using any form of textual descriptions, but just with the identifier. Although this process is not appropriate when users are looking for services, it simplifies the configuration of devices that have to look for specific services when they boot. Pervasive computing environments have to support disconnected operation, hence relying on a central server to maintain and distribute

UUIDs is not a feasible design choice. Although Media Access Control (MAC) addresses have been suggested as a basis for self-generated UUIDs, their use may lead to security problems [GWF+99]. One way to generate UUIDs without relying on any kind of global infrastructure has been proposed as part of the Jini framework [ASW+99].

Service standardization is another important property of service discovery systems. When a service consumer wishes to locate a certain service, she must be able to define in an unambiguous way either the name or the attributes of the required service. More importantly, when the service is located the requester must be able to access its functionality. In order for these problems to be addressed in the context of the inherently heterogeneous pervasive computing paradigm, service descriptions and programming interfaces have to be standardized. There are currently two approaches for standardizing provided services. The first one uses textual descriptions and the second one interfaces[1] of a programming language. Textual service descriptions have to be structured according to a clear and well-defined language in order to facilitate efficient and precise discovery. As an example, UPnP uses the eXtensible Markup Language (XML) which creates structured descriptions, is easily parsed and, as a pure text language, is platform independent. The other approach, adopted by Jini, is to standardize a service with a programming interface that service instances implement. The interface is used to define the high-level protocol that a service consumer can use to access the functionality of the service. This method provides the advantage of shielding underlying implementation details and choices, making possible future changes transparent.

2.4 Pervasive Computing Systems

In the previous sections we have analyzed some of the fundamental properties of pervasive computing along with the technologies that enable its deployment. In order to gain a better understanding of this new computing paradigm and the requirements of its intended target users, this section will focus on presenting practical implementations. Our goal is to present the most representative existing systems, and not to do an exhaustive survey of every pervasive computing project. This analysis is also helpful to infer the associated

[1] The word "interfaces" here is used in its object-oriented meaning.

security implications that will form the foundation of the detailed threat model presented in the next chapter.

2.4.1 Oxygen

The goal of the Oxygen project is to provide a complete pervasive human-centered computing environment. As its name suggests, computation is to be available everywhere around the user responding to natural language commands instead of requiring manual interaction. Hence, Oxygen's two general research areas are speech and vision recognition, for enabling users to communicate with the system as if interacting with another human being, and automation and collaboration technologies to perform the required computation tasks [MIT05]. A general overview of the Oxygen project can also be found in [Der99].

In more detail, the technologies involved in the Oxygen project can be divided into the following five categories:

1. Device technologies. There are two kinds of devices in the Oxygen vision, embedded devices named E21s, and handheld devices, named H21s. E21s are considered to be powerful computation devices playing the role of servers in a smart space. They directly interface typical household appliances, such as air conditioners, TVs, and lighting switches, and expose them to the user through natural language recognition and gesture capturing without offering any specific physical interaction interfaces. H21s are PDAs, universal remote control devices, cellphones and other similar devices that offer mobile access points for users both in and out of the smart space controlled by E21s [MIT05]. H21s can conserve power and computation resources by offloading operations onto nearly located E21s.

2. Network technologies. Networks, which are referred to as N21s by Oxygen, are responsible to create dynamic connections between the mobile and stationary devices of a smart space in order to enable the engineering of collaboration protocols. N21s offer naming, location and service discovery mechanisms.

3. Software technologies. The software support of Oxygen is designed to support change and adaptability. In Oxygen there is no configuration state maintained as every

user creates a new profile as he interacts with the applications of the environment. Applications remain anonymous between user sessions.

4. Perceptual technologies. The main modes of interaction between users and applications in Oxygen are speech and vision instead of keyboards and mice. Speech recognition is enhanced by visual identification of facial expressions, lip movement, and gaze.

5. User technologies. These include automation, collaboration, and knowledge access control. Automation allows users to create scripts and control devices according to their needs. Collaboration is used to accommodate mobility and form links between data collected from the environment and user input. Finally, knowledge access enables users to create their own knowledge bases, customize information access and create semantic connections between data that are meaningful to them.

An important project of Oxygen is the Cricket location-support system [PCB00]. Cricket utilizes radio and ultrasonic signals in order to measure distance and from that to determine location. It allows mobile or static applications to learn their physical location by listening to and analyze information from beacons installed throughout a building. Instead of explicitly tracking user location, Cricket allows devices to learn where they are in a building and leaves the decision of where to advertise this information to the user. This design approach allows a device to know its location, while everybody else including the beaconing system does not.

Example applications developed as part of Oxygen include an answering machine that is able to differentiate between calls based on the contents of incoming messages and forward the urgent ones to the handheld the user is currently using. Also, applications that are able to recognize and answer questions such as "Where did I put my glasses?" since cameras are employed to record and locate physical objects.

2.4.2 EasyLiving

The EasyLiving project by Microsoft Research investigates architectures and the related enabling technologies for supporting invisible computing. The main goal of EasyLiving is

to develop an integrated system architecture for supporting a coherent experience for the users as they interact with the handheld or embedded devices that exist in a smart space [BMK+00]. The devices of the environment are divided into input and output devices. The first part of the architecture consists of software components that encapsulate and represent these devices. Another part of the architecture is the lookup service which registers and catalogs all the software components and exposes their provided services to other components and to applications. In order to support communication between the components, EasyLiving employs a distributed middleware which uses asynchronous messages. XML is used to encode these messages and achieve interoperability between heterogeneous hardware platforms.

Another important element of the EasyLiving architecture is its physical geometry modelling component. By achieving an as accurate as possible model of the smart space, the system allows the physical relationships that exist between users, devices and the environment itself to be used in developing location-aware applications. At this point we must note that EasyLiving provides no other support for storing, interpreting or using any other kind of context[2]. Some example applications of the EasyLiving project include teleporting of active applications between displays of the environment to follow the user as she moves and universal remote controls that locate and provide access to services that are within the vicinity of users.

2.4.3 Cooltown

Cooltown is the pervasive computing architecture designed and implemented by Hewlett-Packard. The core idea behind Cooltown's approach is the representation of every physical real world entity of a smart space with a World Wide Web (WWW) page [KB01], [CD00]. An entity can be an object or device, a person, or even the entire smart place itself, and is identified by a globally unique Universal Resource Locator (URL) that is beaconed by the entity in its vicinity. The URL leads to the page of the entity, and is dynamically updated with context information related to the entity it represents. Therefore, Cooltown focuses on the end user's interaction with objects and the space, accessing collected contextual

[2]This is a general observation, as also noted in [CK00]. Very few contexts other than location are actually used in most current pervasive computing systems.

data and provided services. The two main advantages of Cooltown are its usage model and its adoption of WWW standardized protocols. As a user moves through the space she simply points her PDA to a device she wants to access or wants to learn more about its contextual state. The device's URL is scanned by the PDA and the corresponding WWW page is requested, parsed and presented on the PDA's display to the user. By adopting the web model Cooltown manages to be simple, robust and scalable. The WWW is one of the most well-researched application areas of internetworking, therefore many existing technologies, like web services, can be directly used. Referring back to our discussion on context-awareness, Cooltown follows the passive context-aware computing paradigm. Although each device collects context information about itself, this is only presented to an interested user after a query. It is not used for dynamic behavior adaptation, reconfiguration, or automatic task execution.

Example application scenarios of Cooltown include users that visit new environments and by capturing the space's URL on their PDAs learn of all the services offered. Also, a speaker that wishes to make a presentation to a conference receives the projector's URL with a PDA, visits the corresponding page that implements the projector service's access interface and uploads his presentation files. The projector can then be controlled through the PDA via the web-based exposed user interface [Sta02].

2.4.4 Active Systems

At the AT&T Laboratories Cambridge another, closely related to the ones we have already examined, view of pervasive computing known as *sentient computing* is being researched. The main idea of sentient computing is that systems and applications must react to the user's changing context in order to satisfy her requirements according to the new circumstances [Hop99]. Context-awareness is enabled by deploying sensors, like cameras and microphones, throughout the physical environment that collect status and location data. Hence, the user is not required to physically, for example by using a keyboard, interact with the system; the system is able to collect information about the user's current context and by taking into account her preferences to dynamically reconfigure itself. Consequently, sentient computing as described above follows the active context-awareness computing

paradigm. Practical applications of sentient computing include the Active Badge, Active Floor, and Active Bat systems, referred to collectively here as *active systems*. These, as well as some of their applications, are presented briefly in the following paragraphs.

Active Badge is an indoors location system with a spatial resolution of room scale. A building is equipped with fixed infrared sensors, one in each room. Every person in the building wears a device, like a badge hence the name of the system, which has an infrared transmitter that periodically beacons the device's identifier. Since infrared signals are not able to penetrate walls, a room's sensor is able to receive the beaconed message and register the person that corresponds to the received identifier as being in the same room as itself. This basically means that sensors are configured at installation time with a string that identifies their location, i.e. room number, in the building. All the sensors of a building report the identifiers they receive to a central system server which then is able to know where individual badges are. The Active Badge system has been used in many applications, both of social, like locating colleagues, and technical nature. Examples of the latter include the Call Forwarding [WHFG92] and the Teleporting [BRH94] systems. Call Forwarding uses Active Badge in order to find the location of the recipient of an incoming telephone call, then with a help of a Private Branch eXchange (PBX) system it forwards the call to the destination user's nearest phone. One of the limitations of the Call Forwarding system was that users wanted more control over when calls are forwarded to them based on their context. For example, unexpected calls were not welcome during a meeting [CK00]. The Teleporting system tracks the location of users as they move in the building and maps the user interfaces of the active applications on a user's home machine onto his nearby computer facilities.

The Active Floor system was designed to overcome the requirement of having to carry or attach a badge to each entity that has to be tracked. By creating a pressure-sensitive floor that acts like a precision scale, the researchers were able to track items and people without the need to label them in some way beforehand. The floor is able to detect even slight weight changes, for example like when moving a small object from one desk to another [AJLS97]. However, the main problem of the system is that it is difficult to reach high-level contextual meaning from the low-level floor sensor data. Thus, the Active Floor

system cannot be reliably used as a location system.

Active Bat is the location tracking system that replaced Active Badge. In order to achieve higher position granularity than the room, Active Bat uses ultrasonic signals instead of infrared beaconing. Active Bat receivers are placed at known locations in the building and transmitters are attached to objects or people. As with the badges, Active Bat transmitters have a unique identifier that they transmit with an ultrasonic pulse. The pulses are picked up by the receivers in the environment and by measuring different flight times depending on their relative location to the transmitter, the location of the transmitter is calculated with respect to the known positions of the receivers [WJH97]. This way the Active Bat system achieves an accuracy of about 3 centimeters. In order for this low-level context information to be utilized, a spatial indexing middleware has been developed that allows the generation and dissemination of events [SWH98]. Application developers are able to define the spatial events in which they are interested in and receive notifications when they happen. Thus, the Active Bat middleware supports the development of active context-aware computing applications. The higher resolution location information that the system provides can be used for creating new usage interaction patterns in pervasive computing, for example a user can activate a certain functionality simply by pointing his transmitter to specific place in the smart space [Sta02]. Researchers at AT&T Laboratories Cambridge have used this to create "virtual buttons"; an application developer is able to register a region in the smart space and associate it with a specific action. A user can then use her transmitter to select this region as if using a mouse in a three-dimensional environment and activate the related functionality. Another application is the recording of which user is using which devices. By placing transmitters on objects as well as on users, the system knows which user was using which object at a specific time.

2.5 Computer Security Foundations

In this section we briefly present some fundamental concepts of computer security that are relevant to the research area of this dissertation. We first present secret and public key cryptography, along with digital signatures and public key certificates, as the main building blocks of secure systems. At the same time we introduce the notation that is going to be

used. We then concentrate on the traditional requirements of computer security that are relevant to our work, namely *identification*, *authentication*, and *authorization*[3]. Although the other three conventional requirements of computer security (*confidentiality*, *integrity*, *availability*) need to be addressed as well in a complete security solution, we do not focus our main efforts on them. The reason for this is that if the problems of authentication and authorization have been addressed and some key material have been established between two communicating parties, the protection of the confidentiality and the integrity of the channel with strong secret key cryptography is a trivial problem nowadays. On the other hand we consider availability, which concerns itself with the problem of enabling systems to perform their advertised services in a timely manner, outside the scope of our research area. Availability is a complex requirement which aims to address denial of service attacks from users that have already been authorized to access a service [Gli84], while our focus is on the process of authorization itself.

2.5.1 Secret Key Cryptography

Secret key cryptography allows two parties that wish to communicate with each other (we call them Alice and Bob according to cryptographic tradition[4]) to do so without the fear of someone that is eavesdropping (Eve) on the communication channel learning the contents of the exchanged messages. In order for Alice and Bob to protect their messages they must have agreed beforehand on a quantity, called a secret key and denoted by S, known only to the two of them. The agreement on the secret key must take place over a channel that is guaranteed to be confidential as well as authenticated. After the establishment of the key S, Alice can encrypt a message M to Bob with it. The encryption operation is denoted as $C = E_{S_{AB}}(M)$, where S_{AB} is the secret key shared by Alice and Bob, E is the encryption function, M is the message (also called *plaintext*), and C is the result of the operation (also called *ciphertext*). Alice transmits C to Bob over the unprotected communication channel. Bob decrypts C using a decryption function (which must be based on the same

[3] In this dissertation we treat the terms authorization and access control as synonymous.

[4] However, we will generally avoid extensively referring to entities as "Alice" or "Bob" in this dissertation as this creates confusion regarding the exact nature of the entity, especially in pervasive computing where human users play central roles. When an entity is a human user, a computing device, or a process this will be explicitly defined when required.

algorithm as E) and the shared key; the decryption is denoted by $M = D_{S_{AB}}(C)$. Eve, who does not know the secret key shared by Alice and Bob (S_{AB}), is not able to decrypt C (see Fig. 2.1).

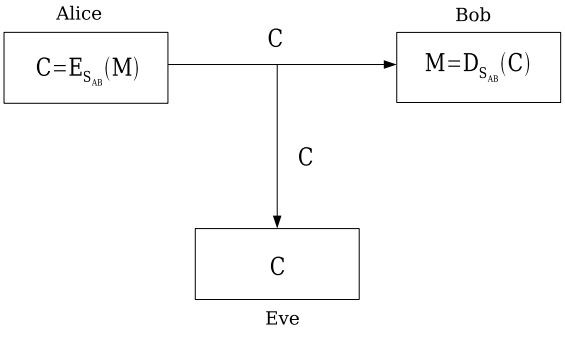

Figure 2.1: Secret key cryptography.

Secret key cryptography algorithms are also known as *symmetric key* algorithms. A lot of them have been proposed in the literature. The American National Institute of Standards and Technology (NIST) has selected Rijndael [DR02] as the Advanced Encryption Standard (AES) in 2000.

2.5.2 Public Key Cryptography

The main problems with secret key cryptography are key distribution and management. As we have seen, the parties that need to communicate have to have exchanged, in the past, a secret key. However, this cannot take place over the channel they use for their normal communications as this is unprotected and an eavesdropper is able to learn any key material exchanged over it. Therefore, they need another channel that is secure for key distribution. Furthermore, symmetric schemes suffer from complex management procedures and increased storage requirements for the shared keys as the number of participants grows. Specifically, in a network of n nodes the total number of different symmetric keys needs to be at least $\frac{n(n-1)}{2}$, assuming that each pair of nodes shares a common key. One of the most important contributions to the problem of establishing and managing keys is public key cryptography, initially explained by Diffie and Hellman [DH76]. Their design addressed a subset of the intended properties and it was focused on key agreement between two parties.

Rivest, Shamir and Adleman soon afterwards developed the RSA scheme [RSA78] which was the first full implementation of a public key system.

Public key cryptosystems rely on two different keys for each entity, known as the *key pair*. An entity A (possibly Alice) generates a key pair according to a specific algorithm, and the result consists of the public key K_A and the private key K_A^{-1}. A then publishes the public key and keeps the private one secret. When B (for example Bob) wants to sent a message to A he has to find K_A and then use it along with the encryption algorithm E to encrypt a message M; we denote the encryption operation by $C = E_{K_A}(M)$. The resulting ciphertext C is then sent to A who uses her private key K_A^{-1} and the decryption algorithm D to decrypt it and get M, the decryption process is denoted by $M = D_{K_A^{-1}}(C)$, see Fig. 2.2.

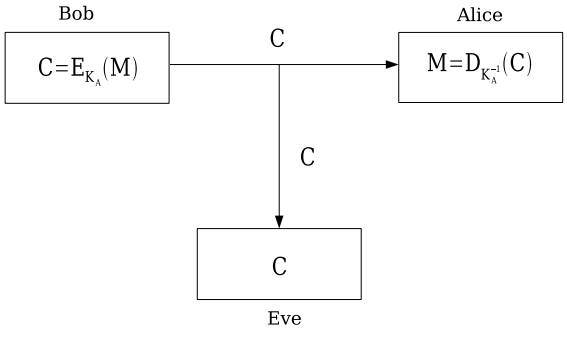

Figure 2.2: Public key cryptography.

Obviously, in public key cryptosystems, also known as *asymmetric key* cryptosystems, it should not be possible to calculate the private key from the corresponding public key. Another property that has to be satisfied is that for all possible messages M the following has to hold: $D_{K_A^{-1}}(E_{K_A}(M)) = M$. The main advantages of asymmetric cryptosystems over symmetric ones are that two entities are able to exchange confidential messages securely and verify the validity of these messages provided they know the other party's public key in some trusted manner. Also, there is no requirement for keeping the public keys secret.

2.5.2.1 Digital Signatures

Digital signatures are constructs of public key cryptosystems that provide *non-repudiation*. When Alice sends a message to Bob she uses her private key to sign it (we denote a signed message M with Alice's private key K_A^{-1} as $[M]_{K_A^{-1}}$). Bob, or any other party that knows Alice's public key K_A, is able to verify the message and be sure that it has been signed with Alice's private key and that it has not been modified in transit. In order to avoid creating digital signatures on arbitrarily large messages, a process which is particularly computationally expensive, it is common practice to pass the message as input to a cryptographic one-way hash function and sign the output message digest.

2.5.2.2 Public Key Certificates

Although public key cryptosystems make key management simpler in environments with many participating principals, there is the problem of acquiring the correct public key of the recipient principal. In our example above, when Bob wants to send a message to Alice he needs to be sure that he is using Alice's public key to encrypt the message and not the public key of another principal which he falsely believes is Alice. Public Key Infrastructure (PKI) tries to prevent attacks in which a malicious user publishes the public key of a key pair he generated (and therefore has the corresponding private key) as the public key of another user. The main components of PKI systems are Certification Authorities (CAs) that generate and digitally sign statements known as *public key certificates* which bind an entity's name (or another preferably unique and unambiguous identifier) to that entity's public key. A certificate also includes expiration data, along with other information. Returning to our example, Bob can now verify Alice's public key certificate, assuming that he knows and trusts the CA which issued Alice's certificate, and be sure of the mapping between Alice's identity and her public key. The advantage of public key cryptosystems over secret key cryptosystems is made clear in a PKI environment; a principal needs to know only the CA's public key in order to verify the certificates of other principals. There is no need of key exchange and storage between each pair of principals that need to communicate. For scalability reasons there is no single CA that issues certificates for everybody. Instead there is a hierarchy of CAs. The root CA issues public key certificates for lower level CAs

which in turn certify user keys.

Revocation of public key certificates is required when the information contained in a certificate is no longer valid, possibly because of key material compromise, even though the expiration period has not elapsed. There have been several proposals in the literature to address the problem of revocation, like Certificate Revocation Lists (CRLs) [AF99], delta CRLs [HFPS99], the Online Certificate Status Protocol (OCSP) [MAM+99], short term certificates [Riv98] and windowed revocation [MJ98]. These methods provide different tradeoffs in terms of stored and transmitted information, window of vulnerability and online availability requirements. Consequently, there is no single method applicable to all scenarios; the design of a revocation mechanism should be driven by the application environment it is supposed to support [MR00].

Although we will thoroughly analyze PKI-based solutions to security management in following sections, it should be noted here that PKI does not solve all problems. For example, the requirement that a CA principal must be trusted by all system participants cannot always be satisfied in all application scenarios. Also, the introduction of a CA creates a single point of attack and failure in the system; if it gets compromised all secure communication channels can be compromised as well.

2.5.3 Identity and Authentication

The term "identity" has been used with many different meanings and has been overloaded in the computer security literature, a fact that makes it controversial. According to Anderson [And01], identity refers to a correspondence between the names of two principals signifying that they refer to the same person or equipment. For example, Bob's identity can be defined as the correspondence of Bob being the same person acting both as Alice's and Charlie's manager [And01]. A "principal" is defined as an entity that participates in a security system, such as a role, a person, a computer, or a smartcard. Of course this largely depends on the definition of "security system". For example, in security systems that are based on public key cryptography, principals can be public keys. In this dissertation we will generally use the term "principal" as synonymous to the term "software entity" and we will be providing more details when they are required. As it is suggested by Ford,

authentication can be defined in terms of a principal and the identity it claims to have [For94]: "authentication relates to a scenario where some party has presented a principal's identity and claims to be that principal". Therefore, we can view an identity as the name of a principal that is presented during a security process in the system, and the corresponding principal can prove that it legitimately holds it through the process of authentication.

2.5.4 Authorization

Lampson [Lam74] defines authorization as the process according to which a reference monitor determines whether a principal is allowed to perform a certain action it requests on a resource. Authentication is performed before authorization in order to verify that the requesting principal legitimately holds the identity it claims to have when it requests access to a resource. Moreover, the process of authorization depends on an *authorization (or access control) policy* which refers to the access rules enforced by the reference monitor. Based on the above, we can divide the process of authorization as performed by the reference monitor into the following three steps (summarized in Fig. 2.3):

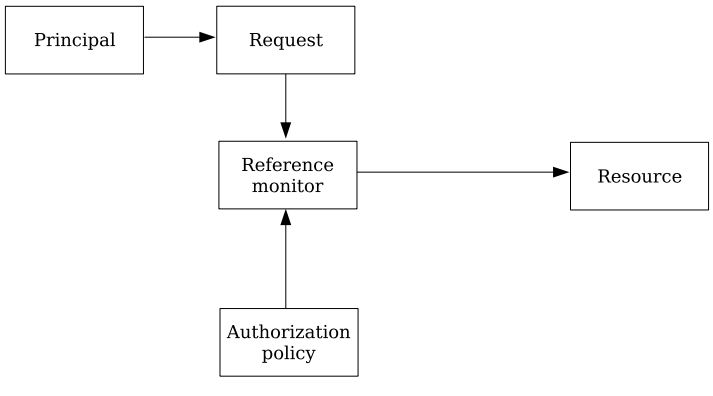

Figure 2.3: Authorization model.

1. Determine the identity of the principal that makes an access request.
2. Authenticate the identity of the requesting principal.
3. Determine whether the access request is allowed or not based on the result of the authentication (from step 2) and the access control policy.

The traditional method for defining access control policies is Access Control Lists (ACLs). They are used by the reference monitor to determine the appropriate access rights to a given resource given certain aspects of the requester, principally its identity. An ACL is usually implemented as a table containing entries that specify the rights a principal has over specific system resources.

2.6 Computer Security Systems and Models

This section presents traditional computer security systems that address the problems of authentication and authorization in networking environments. Our examination focuses on their basic characteristics and the reasons they fail to satisfy the requirements of authorization in pervasive computing. Although we examine these systems with regard to the previously identified requirements, we do not expect them to fully satisfy all the pervasive computing requirements since they were not designed for such a purpose.

2.6.1 Role-Based Access Control

The Role-Based Access Control (RBAC) model [FK92] provides the ability to specify and enforce domain-specific access control policies and simplifies the process of authorization management in computing environments with a great number of users. This is achieved by associating roles with access permissions and users with roles. In essence, RBAC defines all access rights through roles. A role is an entity that acts as a collection of permissions. Users are assigned to roles, usually by an administrator on a central system, and receive access permissions only based on the roles they have been assigned to (illustrated in Fig. 2.4).

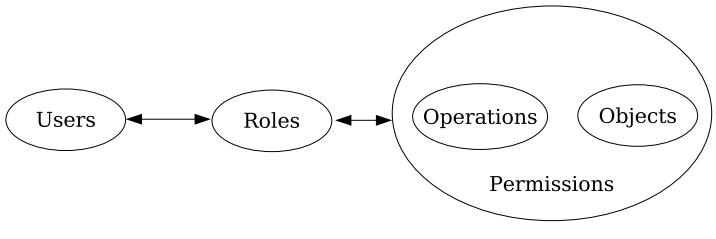

Figure 2.4: Basic Role-Based Access Control (RBAC) model.

Ferraiolo and Kuhn, extending a large body of previous research work, proposed the first generalized RBAC model which formally described the sets, relations and mappings required to define roles [FK92]. They defined the terms *user*, *subject*, *object*, *operation* and *permission* as follows:

- A *user* is a human operator of a computer system.
- A computer process that acts on behalf of a user is called a *subject*.
- An *object* is a resource available on a computer system, for example a file, an entry in a database, or a printer.
- An *operation* is defined as an active process invoked by a subject. It should be noted that a subject and an operation are distinct entities; A user can invoke a subject which can then invoke operations itself.
- A *permission* is an authorization to perform some action on a computer system and is defined as a combination of an operation and an object.

According to their model, RBAC requires the three following rules:

1. Role assignment. A subject can perform an operation only if it has been assigned to a role.
2. Role authorization. A role that a subject has been assigned to must be authorized for the particular subject. The rules of role assignment and role authorization ensure that a user can only have active roles she is authorized to have.
3. Operation authorization. An operation can be performed by a subject only if the specific operation is authorized for the subject's active role (or roles).

An important feature of RBAC is that it provides a mechanism for preventing users from assuming roles that are conflicting. Separation of duty allows two roles to share user members, but no user is allowed to have both roles active at the same time. This is useful when a system needs to ensure that a single user does not acquire too much authority and therefore is able to undermine the operation of the system. For example, a system may

allow a user to assume the roles "Treasurer" and "Payroll Officer" at different times but not both simultaneously.

The advantages of RBAC when compared to other access control mechanisms such as ACLs are obvious. An organization can define a set of roles that can remain unmodified from that point on, and can define volatile permissions for these roles according to the functions that need to be performed as part of its ongoing operation. Users are then assigned to roles receiving the related access permissions. These assignments do not have to be permanent, but they follow the organization's human resources management procedures. For example, if an employee's job changes then only two changes have to be performed; one to remove the association between the employee and the employee's current role, and one to add a new association between the employee and her new role. On the other hand in the ACL model all available computing resources have a list of the users that are authorized to access them (see Table 2.1). In environments with many users the process of enumerating each user's identifier along with the permissions that this user has comes at a great administrative cost. This cost increases even more when the set of participating users changes frequently. As we have already seen, pervasive computing environments are inherently dynamic and therefore not compatible with the use of ACLs as a security mechanism.

Table 2.1: Example Access Control List (ACL).

Resource	**User permissions**
File-001	Alice: read/write, Bob: read
File-002	Alice: read/write/execute, Eve: read
Printer-001	Alice: read/write, Bob: read/write

Sandhu et al. have extended the core Ferraiolo-Kuhn RBAC model to four different conceptual models [SCFY96]. $RBAC_0$ is their base model, which is equivalent to the core Ferraiolo-Kuhn model, and defines the minimum requirements for any system that aims to support RBAC. $RBAC_1$ adds the concept of role hierarchies to $RBAC_0$, which basically allows roles to inherit permissions from other roles. $RBAC_2$ adds constraints to $RBAC_0$ that impose restrictions on acceptable configurations of the different components of RBAC.

Finally, RBAC_3 includes the basic RBAC_0 model as well as RBAC_1 and RBAC_2. The core Ferraiolo-Kuhn RBAC model and the RBAC_0 model are composed of the following components (a graphical representation of which is shown in Fig. 2.5):

- A set of users: U
- A set of roles: R
- A set of subjects: S
- A set of objects: O
- A set of operations: OP
- A set of permissions: $P = \mathcal{P}(OP \times O)$[5]
- User to role assignment: $UA \subseteq U \times R$
- Permission to role assignment: $PA \subseteq P \times R$
- The mapping of a role to a set of users: $assigned\_users(r : R) \rightarrow \mathcal{P}(U)$, or more formally: $assigned\_users(r : R) = \{u \in U \mid (u, r) \in UA\}$
- A function mapping a subject to the subject's associated user: $subject\_user(s : S) \rightarrow U$
- A function mapping a subject to a set of roles: $subject\_roles(s : S) \rightarrow \mathcal{P}(R)$, or more formally: $subject\_roles(s : S) \subseteq \{r \in R \mid (subject\_user(s), r) \in UA\}$
- A function mapping a role to a set of permissions: $permission : R \rightarrow \mathcal{P}(P)$, or more formally: $permission(r : R) = \{p \in P \mid (p, r) \in PA\}$
- Access authorization: A subject s can perform an operation op on object o only if there exists a role r that is included in the subject's active role set and there exists a permission p that is assigned to r such that the permission authorizes the performance of op on o: $access : S \times OP \times O \rightarrow BOOLEAN$, $s : S$, $op : OP$, $o : O$, $access(s, op, o) \Rightarrow \exists r : R, p : P, r \in subject\_roles(s) \wedge p \in permission(r) \wedge (op, o) \in p$

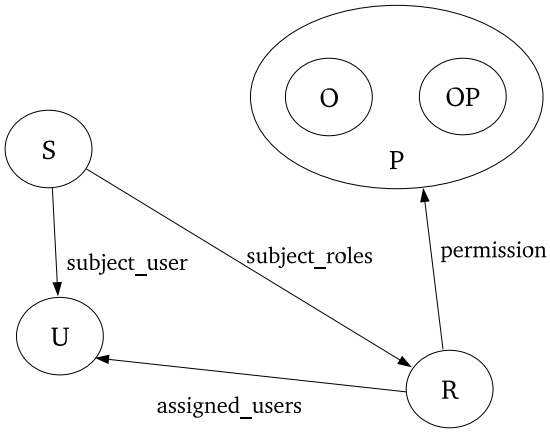

Figure 2.5: The core Ferraiolo-Kuhn RBAC model.

The model has a special kind of permissions, called *administrative permissions*, which allow the roles that have them to modify the sets U, R, and P as well as the relations UA and PA. The term *authorization management* refers to the problem of managing these administrative permissions and their assignment. Although there have been many solutions proposed in the literature to address this problem in RBAC systems, all of them rely on centralizing authorization information on a server that is always assumed to be online and readily accessible in order to provide the needed data. Furthermore, a human security administrator is required to manually configure the system and assign permissions to roles and roles to end users. This centralized management process is supported by two different methods that facilitate access to applications and services [FKC03]. According to the first one, named *push*, a user is authenticated by the central authorization server and obtains some sort of credential to access applications. The user then presents (or *pushes*) these credentials to the applications as proof of authorization (illustrated in Fig. 2.6).

The second method, called *pull*, requires applications to perform user authentication and query the central authorization server about the user's permissions, *pulling* them for local access (shown in Fig. 2.7). Thus, the functions of authentication and authorization are performed by two different conceptual entities in the pull method; authentication is done by applications and authorization by the central server.

Of course, before users are able to access services and applications the security ad-

[5] $\mathcal{P}(A)$ denotes the power set of A, that is the set of all possible subsets of A.

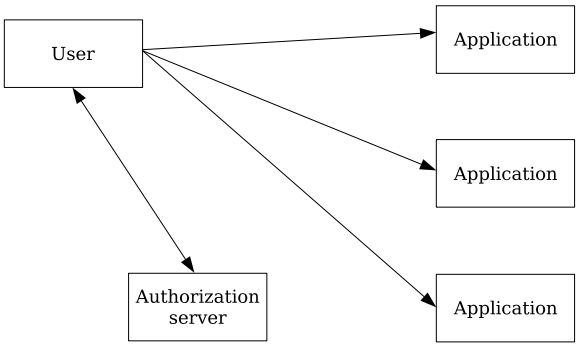

Figure 2.6: Push authorization method.

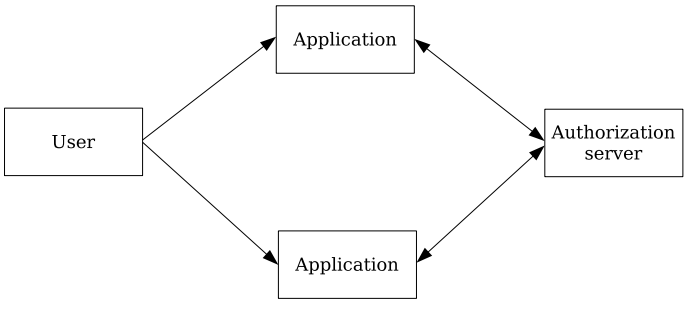

Figure 2.7: Pull authorization method.

ministrator must configure the required permissions, create associations between roles and these permissions, and finally assign roles to users. Although RBAC is far more flexible in management and in detail of control when compared to other access control management mechanisms, such as Mandatory Access Control (MAC) and Discretionary Access Control (DAC), it still requires centralized management via an online server and direct human intervention, as proposed by the National Institute of Standards and Technology (NIST) RBAC specification [FSG+01]. According to MAC, access to resources is defined based on the specific resource's *sensitivity* as it is defined by a central authority. Therefore, RBAC relates to MAC in the sense that end users have no control over the permissions defined by the security administrator and enforced by the central authorization server. On the other hand, DAC permits the granting and the revocation of permissions to be left to the discretion of the individual end users of a system. A user that controls certain resources is able to assign and revoke rights for them without the need to contact a central server, or be authorized to do so by an administrator. Consequently, the NIST proposed RBAC standard is a *non-discretionary* access control model. Role assignment remains a centralized operation and requires the manual intervention from an administrative authority. This requirement as well as the requirement of having an always accessible online central server prohibit RBAC to be directly used in securing pervasive computing environments.

2.6.2 X.509 and the X.509 Attribute Certificate Profile

As we have already briefly discussed, a Public Key Infrastructure (PKI) is a system that allows entities to recognize and securely bind public keys to global identifiers. The central entity of a PKI is the Certification Authority (CA) which has a private/public key pair, as does any entity that participates in the system. One of the main assumptions of PKI is that all participants have the CA's public key and that they trust the CA as an authority to issue bindings between public keys and global identifiers. In order for an entity to join the system it must present its global identifier (usually a Domain Name System (DNS) name) and its public key to the CA. The CA then verifies that the entity is really the one it claims to be[6] and that it has the corresponding private key by issuing a challenge encrypted with

[6]Usually with some offline method, like requesting a national ID card. In reality few CAs go through this procedure and just rely on an email exchange as an alternative.

the presented public key. If all tests pass successfully the CA signs a statement asserting that the presented public key belongs to the presented global identifier of the entity. Such signed statements are called X.509 Public Key Certificates (PKCs) or Identity Certificates (ICs) [ISO01]. When two participants wish to communicate securely they exchange PKCs and each one verifies the other's certificate with the public key of the CA they both have and trust to issue to such signed statements. Upon verification they are both sure about the identity of the other and can then use a key establishment protocol to derive key material for creating a secure communication channel. One of the main characteristics of PKI is that non-CA (for example end user) keys are not allowed to issue certificates and certify bindings between names and keys. This obviously leads to scalability problems. In order to overcome this limitation and scale to an Internet-wide system, X.509 follows a hierarchical trust model. For example, if we have a two-level structure at the top would be the central CA that signs certificates for the CAs of the lower level. Each second level CA would be responsible for a different administrative domain, such as an organizational department. Users are certified by the second level CA that corresponds to the domain they belong to. When a user of a particular domain wants to authenticate himself to a user of another domain he must present two certificates. One signed by the top CA that certifies the CA of his domain and one signed by his domain CA that certifies the binding between his identifier and his public key. The receiver must verify both certificates; the first one with the public key of the root CA that she has by being a system participant, and the second one with the public key contained in the first certificate she has already verified. This is called a *certificate chain* and can span an arbitrary number of CA hierarchy levels.

The X.509 authentication infrastructure offers little help in supporting complex access control decisions. Although it can be used to establish the identity of an entity and provide authentication services by employing a centralized architecture, most systems are not interested in the name or the identity of an entity. The real requirement is to know what operations on which resources an entity is authorized to perform. To address this problem with the standard X.509 architecture we need to enumerate all the identifiers of the participating entities in a system, create ACLs for every resource in the system and for every supported action and bind identities to permissions. Obviously when we have

to manage authorization information in large systems with great numbers of users this approach fails to scale. X.509 Attribute Certificates (ACs) were introduced to address this problem [FH02]. They bind a user's identifier with one or more *privilege attributes*, which are defined as attribute type/value pairs. ACs are issued by Attribute Authorities (AAs) that play they same role as CAs, but focus on authorization instead of authentication certifications. The X.509 AC infrastructure is called a Privilege Management Infrastructure (PMI) and it can also follow a hierarchical trust model, as a normal PKI. The root element of a PMI is called the Source of Authority (SOA). However, PMI still relies on the existence of a PKI, since an AC does not contain a public key. When an entity wants to perform an operation on a resource it has to present to the reference monitor both a PKC and an AC. The reference monitor authenticates the requester by verifying its PKC, and then checks to see if the name included in the PKC is the same as the name on the AC. If they are the same and the AC can be verified as well it then accepts the privilege attributes of the AC as authorization credentials. There are several ways to bind identifiers and attributes in X.509 [PS00], one of these is shown in Fig. 2.8.

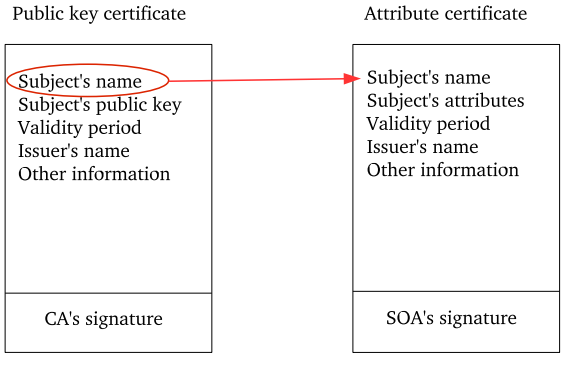

Figure 2.8: Public key certificate and attribute certificate binding.

One of the most difficult problems in X.509, and indeed in all certificate-based security systems, is revocation. We have previously mentioned that revocation refers to the need of withdrawing a certificate before it expires. Since certificates constitute data that are supposed to be public it is difficult to inform all parties that have acquired them about their possible revoked status. The main requirement of revocation is speed; that is the time between the revocation command and the last use of the revoked certificate should

be as small as possible. The two main solutions to this problem are Certificate Revocation Lists (CRLs) and short term certificates. CRLs are databases that contain the revoked certificates and are distributed by CAs. When an entity wishes to verify a certificate it must contact the CA that issued it, request the CRL and make sure that the certificate it was presented with is not listed in it. Of course this approach requires constant online access to every CA that issues certificates for a given system. Short term certificates are simply certificates that are valid for short periods of time, usually between ten minutes and a few days. When an entity wishes to use a certificate to authenticate or authorize itself it checks to see if it has expired. As long as it remains valid it can be used normally. Otherwise, the CA (or AA) that issued the certificate in question must be contacted and a new one must be acquired.

X.509 PKI and PMI models are not able to address the problem of authorization in pervasive computing environments, mainly because they were not designed for such a purpose. Their main disadvantage is the requirement of a globally trusted central server that is assumed to always be online and readily accessible. In pervasive computing we have usage scenarios that take place in all possible locations that may or may not have access to an Internet gateway, hence the ability to support disconnected operation is crucial. Another limitation of this approach lies in the use of identities as a basis for building trust and authorizing service requests. PKI and PMI credentials bind the owner's public key to a name. As we move to pervasive computing authority domains, naming becomes increasingly locally scoped. Therefore, a name may not have meaning to a different domain than the one that it was certified in. Scalability problems also exist in PKI when authorization decisions are based on identities in dynamic environments since enumeration of all the participating entities is nearly impossible.

2.6.3 Pretty Good Privacy

Pretty Good Privacy (PGP) was originally developed to provide security services for email communications. It is based on public key cryptography and defines its own key management system and certificate format. The goal of PGP is to assure an email user that a public key she has for a particular email address indeed belongs to the corresponding

person. The PGP key management system is based on the relationships between key owners, rather than on a single globally available and trusted infrastructure, as is the case with X.509. Another important difference with X.509 is that every PGP user is allowed to issue public key certificates for other users and certify their keys with his own acting as a certifying authority. PGP certificates basically certify that a user believes to the best of his knowledge that the subject's key and name included in the certificate indeed match. To a third party that must decide on the validity of the subject's key, this signed assertion worths only to the level that the issuer is trusted as a sincere and competent introducer [Zim95]. In PGP each user maintains a *key ring* that contains the keys and the associated identifiers (names and email addresses in this case) of other users. When a key is entered in the key ring the owner must decide on a *trust level* for the new key. The trust level is basically a rating on how much the key ring owner trusts PGP key certificates issued by this new key. The lower the trust value the more key certificates are necessary to validate a given key. Table 2.2 presents the different trust levels used in PGP and their meanings.

Table 2.2: PGP trust levels.

Trust level	**Meaning**
Untrusted	Public key certificates issued by this key are ignored.
Marginal	At least two keys of this trust level have to certify a third key to make it valid.
Complete	At least one key of this trust level has to certify another key to make it valid.
Ultimate	This key has been created by the user herself, therefore she controls the corresponding private key. All keys certified with it are valid.

Based on this key management model PGP creates a "web of trust". Certificate revocation in PGP is handled implicitly and not with any specific mechanism. When a user wishes to revoke a certificate issued previously she posts the revocation notice in a public place, like a website. It is the responsibility of potential users of the certificate to check if it has been revoked or not. In the X.509 PKI model only CAs have the ability to revoke certificates by signing a CRL with the private key used to issue the revoked certificate in the first place. On the other hand, in PGP the revocation notice must be signed by the individual end user that issued the certificate without the need to rely on any central en-

tity. However, this flexibility comes with an administrative cost. X.509 CRLs contain the complete set of revoked certificates from a particular issuing CA. Therefore, when an X.509 PKI participant receives a CRL it becomes fully synchronized with all the valid certificates of the particular CA with a single operation. In PGP every time a user wishes to verify a single certificate she must check the issuer's public repository for possible revocation notices.

PGP has not been designed for providing authorization but authentication services. It relies on a combination of the direct and hierarchical models for managing security relationships. Therefore, it does not have the requirement of a central globally trusted online server in order to operate and establish secure communication channels between users. Every participant is allowed to issue identity certificates for every other participant and based on assigned trust levels these certificates are assumed to be valid or not. When it comes to addressing the problem of authorization, PGP has the same limitations as X.509. Users must be enumerated and ACLs for their identifiers must be created and maintained for all available resources and operations.

2.6.4 Kerberos

Kerberos is a distributed authentication service that allows a client running on behalf of a principal to prove its identity to a verifier (for example an application server), without sending data across the network that might allow an eavesdropper or the verifier to subsequently impersonate the principal [KN93]. Although there have been many extensions and different approaches, the original Kerberos architecture relies on symmetric cryptography and a central online server to offer authentication services in a networking environment. Each valid end user is given a secret key, in the form of a password, which is shared with the main Kerberos authentication server called Key Distribution Center (KDC). When a client wishes to establish a secure channel with a service she contacts the KDC and requests a Ticket-Granting Ticket (TGT). The TGT is sent to the user encrypted with the secret key they both share. For each service the user wants to use she has to first contact the Ticket Granting Server (TGS)[7] and obtain a service-specific ticket by presenting

[7] Usually the KDC and the TGS are on the same physical computing device.

her TGT [KN93]. The user can then contact the service provider and use this ticket as an authentication credential. Both the TGT and the normal tickets have long expiration periods, therefore the user can reuse them many times without having to go through the whole protocol frequently. The whole dialog process is summarized in Fig. 2.9. Kerberos can also be used to offer *cross-realm* authentication services by allowing the TGS of an administrative domain to be a normal principal in another domain [TKS01]. Hence, users of the second domain can obtain tickets for the first domain.

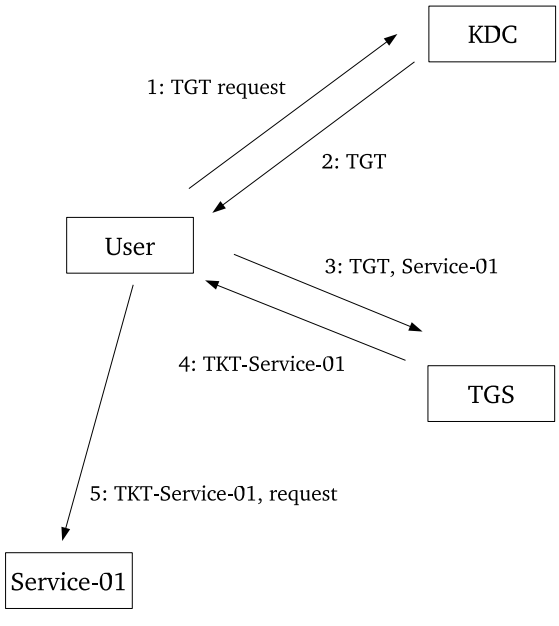

Figure 2.9: The Kerberos authentication protocol.

One of the disadvantages of the original Kerberos architecture is that it does not support authorization. Service providers are expected to reach access control decisions based on other mechanisms, for example using ACLs stored locally. This limitation has been partially addressed by Westerlund and Danielsson that allow some authorization data to be included in Kerberos tickets [WD01]. The main deficiency of Kerberos however when applied in pervasive computing is its reliance on the server (or servers) that host the KDC and the TGS services. These not only have to be available constantly because without their presence no secure connections can be established, but also represent a single point of failure in the system. Furthermore, they require extensive human intervention for the

registration of users and services. Therefore, Kerberos is not directly compatible with the decentralized and unobtrusive nature of pervasive computing.

2.6.5 KeyNote

KeyNote [BFK99] is a security system specifically developed to address the problem of authorization management. It was designed as a successor to an earlier system called PolicyMaker which introduced the term *trust management*. Blaze et al. define trust management in [BFL96] as "a unified approach to specifying and interpreting security policies, credentials, and relationships that allow direct authorization of security-critical actions". As we have seen in previous paragraphs, traditional security management systems, such as X.509 and Kerberos, address only the problem of authenticating the identity of an entity. In order to support authorization in these systems an external mechanism must be developed and exposed to service providers. Trust management avoids the use of identities and instead binds access rights directly to public keys. Verification of identities is considered to be a distinct problem from the problem of assessing trust. This basically means that entities are named by the public key of a key pair they have generated themselves and therefore have access to the corresponding private key. KeyNote credentials are called *authorization or capability-based certificates*, and are used to delegate permissions directly from the key of the issuer (or the *authorizer* in KeyNote terminology) to the key of the subject (or the *licensee*) enabling access control decisions between strangers without the need for a universally Trusted Third Party (TTP). Delegation is allowed to any depth; the subject can further delegate all or a subset of the permissions given to him by the issuer to another key. Credentials which have the special value $POLICY$ in the authorizer field are considered to be locally trusted, are not signed, and play the role of authorization policies. Such *assertions* are defined using an expression language that provides constructs for the specification of complex policies. An entity may grant to another entity access to a service it provides if the provider can determine that the credentials of the requester satisfy the local policies that protect the service. The KeyNote inference engine is called by an application and is given as input the list of credentials the requester presented, the local access control policies and an *action environment* which is basically a set of attribute-value pairs

generated by the calling application. These pairs are used to represent the application's security requirements. The output of the inference engine is either *true* or *false*, depending on whether the request is authorized or not. The process is illustrated in Fig. 2.10. Revocation of credentials in KeyNote is performed through the use of short expiration periods.

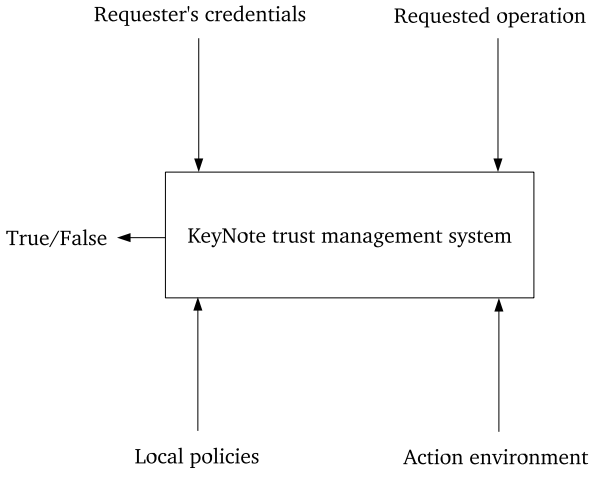

Figure 2.10: The KeyNote trust management system.

One of the main disadvantages of KeyNote is that it does not support negative authorizations which are important for providing solutions to separation of duty problems. Furthermore, KeyNote does not include the functionality of verifying that the credentials presented by a requester actually belong to the requester. This is usually performed with a challenge-response protocol that allows the verifier to ensure that the requester actually possesses the private key that corresponds to the public key used to identity the requester in the credentials. KeyNote leaves this crucial feature to be implemented by the service provider. Generally we believe that the decentralized approach to authorization management that is advocated by KeyNote is compatible with pervasive computing. Users can define their own access control policies for their devices, and give credentials to other users without the requirement of having to contact a central server. Although the concept of the action environment could be used to provide minimal static context information from the application to the inference engine, it is not able to support dynamic context-awareness.

Moreover, we believe that the delegation of a specific set of access permissions to a key is not appropriate for the pervasive computing paradigm in which the initiation of new services is frequent and the number of users particularly large. According to our view, a user that wishes to provide a new service should not be burdened with the additional administration overhead of initiating delegation chains and distributing the credentials that define the access rights for the new service.

2.6.6 SPKI/SDSI

Other capability-based efforts similar to KeyNote that attempt to provide solutions to decentralized authorization through the use of public key cryptography include SPKI [EFL+99] and SDSI [RL96]. The design of the Simple Public Key Infrastructure (SPKI) is similar to that of KeyNote; the issuer of an SPKI credential encodes the permissions she wishes to delegate to a subject using *s-expressions*, which are basically a set of constraints for application-dependent issuer-defined values. As an example consider the following s-expression from the SPKI/SDSI Request For Comments (RFC) [EFL+99] that allows all kinds of access (read, write, delete, etc.) to the directory */pub/cme* of the FTP host with DNS name *ftp.clark.net*:

(ftp (host ftp.clark.net) (dir /pub/cme))

The Simple Distributed Security Infrastructure (SDSI) introduced the concept of local namespaces that allow a principal to define his own locally scoped names in order to refer to other principals. Certain principals that are referred by others with the same name can be used as trusted intermediaries to link multiple local namespaces and create global names. For example, George can define a name "Fred" in his own namespace [EFL+99]:

George: (name Fred)

Fred in turn can also define a name in his namespace, for example:

Fred: (name Sam)

Now George can refer to this same entity as:

George: (name Fred Sam)

This allows communities to be built in a distributed manner following a bottom-up approach. As in KeyNote, all authority comes from the issuer of a credential and can be delegated. A requester provides to a service provider a set of credentials that form a chain from the public key of the issuer to the public key of the requester. If the provider is able to validate the chain, verify all the credentials, and ensure that they satisfy the local authorization policies then the request is granted. Again, revocation is handled with short validity periods for the issued credentials. SPKI and SDSI have merged their design efforts to produce SPKI/SDSI 2.0 that provides a coherent treatment of certificates, names and encoding of policies [CEE+01], [RL96].

2.6.7 Trust Establishment

The Trust Establishment (TE) system developed by IBM Research aims to provide security for e-commerce transactions by employing a public key certificate design based on X.509 [HMM+00]. TE allows certificates to be issued by various organizations certifying subjects with certain attributes. These certificates are then used by service providers to map requesters to roles. Thus, the problem of trust establishment is defined as assigning roles to subjects and then carrying out RBAC for authorization. The TE system includes a Trust Policy Language (TPL) which is based on XML and is used to write policies that specify what attributes, certified by which issuers are required in order for a subject to be mapped to a specific role. An inference trust engine interprets TPL policies and the attribute credentials of requesters assigning them to access roles. These roles are then used according to a traditional RBAC approach to reach access control decisions. The whole procedure is illustrated in Fig. 2.11. TPL can also used to define access control policies of what a role is permitted to do. Its basic XML tags are <GROUP> that defines a role/group[8] in an organization and <RULE> that is used to define the requirements for membership to the specific role. A requester needs to satisfy any one of the rules to become member of the role. Each <RULE> element can include <INCLUSION> and <FUNCTION> tags that indicate constraints for the attributes of the required credentials.

[8]TPL uses the terms *role* and *group* interchangeably.

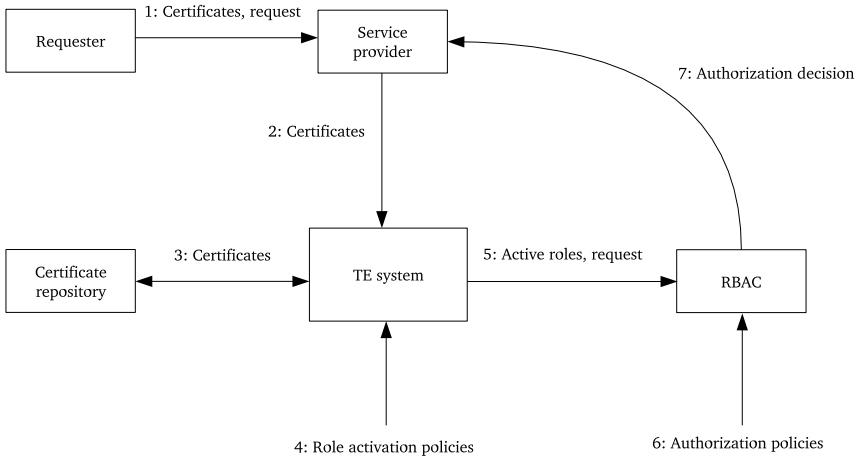

Figure 2.11: The Trust Establishment system.

Since the system allows the issuing of negative policies that specify authorization restrictions, i.e. it is not *assertion monotone*, it also provides a mechanism for collecting all of the credentials related to a principal. This is accomplished with a component called the *certificate collector*, which is designed for collecting missing certificates. The requester specifies along with his request to the provider the location of repositories that hold more certificates issued to him. The provider then uses an automatic certificate repository retrieval agent for collecting all of the credentials. These credentials can be used to verify whether the issuers of the requester's credentials are qualified to issue certificates for the related attributes. However, a malicious requester may avoid to submit the repository location of certain certificates to a service provider with the goal of accessing a role with greater authority. The authors having identified this shortcoming developed Definite TPL (DTPL) as a subset of TPL that no longer supports negative authorizations. Another disadvantage of TE is its inability to support the passing of variables from the calling application to the inference engine. Thus, it cannot support authorizations based on dynamic information such as environmental context data. The main drawback of TE when applied to the paradigm of pervasive computing is that it has been designed for networking environments with a complete infrastructure. It requires repositories of certificates instead of allowing users to possess and carry with them their own credentials.

2.6.8 PERMIS

The Privilege and Role Management Infrastructure Standards Validation (PERMIS) project has been designed to address the authorization requirements of widely distributed networking environments. PERMIS has two main components; the privilege allocation subsystem and the privilege verification subsystem [CO02]. The former is used to allocate privileges to users and the latter to authenticate and authorize them. Specifically, an administrator issues through the privilege allocation subsystem X.509 Attricute Certificates (ACs) to end users which are stored in publicly accessible Lightweight Directory Access Protocol (LDAP) servers. These are used by the privilege verification subsystem to offer RBAC-based authorization services to various applications. Furthermore, a user must also be authenticated according to an application specific method. PERMIS itself is authentication agnostic, therefore a user can present an X.509 identity certificate if a PKI is in operation, a traditional username/password pair, or any other way an application supports to verify her identity [CO02]. PERMIS uses XML in order to specify authorization policies by defining a Data Type Definition (DTD)[9] with the following components [CO02]:

- *SubjectPolicy* specifies the subject domain, i.e. the domain that user must belong to in order to access the provided resources.
- The component *RoleHierarchyPolicy* specifies roles and their hierarchical relationships.
- *SOAPolicy* is used to specify which SOAs are trusted to allocate which roles.
- *RoleAssignmentPolicy* specifies the roles that can be allocated to subjects and the SOAs that are trusted do so. It also specifies whether a subject is allowed to delegate the allocated role and expiration information regarding the allocation.
- The component *TargetPolicy* specifies the domains to which the current policy applies.
- *ActionPolicy* specifies the actions that service providers offer, as well as the types of parameters that these actions accept.

[9] A DTD is in essence a meta-language used to define the rules required to write XML documents. In the case of PERMIS these documents are authorization policies.

- *TargetAccessPolicy* specifies which roles have permission to perform which actions on which resources and under which conditions. These conditions can be used to encode contextual information to PERMIS policies.

When a client makes an access request to a service provider, it has to go through an application gateway which has access to the PERMIS system. There are two main components; the Access Enforcement Function (AEF) and the Access Decision Function (ADF). The AEF is responsible to authenticate the client that makes the request based on the utilized authentication mechanism. The authenticated client identifier is then passed to the ADF component which uses it in order to retrieve from the LDAP repositories the relevant role ACs. Moreover, AEF passes the access request to the ADF which also retrieves the relevant policy ACs. By parsing the XML policy information encoded in the policy ACs and the retrieved role ACs the ADF component then reaches an authorization decision for the requested operation. The system is summarized in Fig. 2.12.

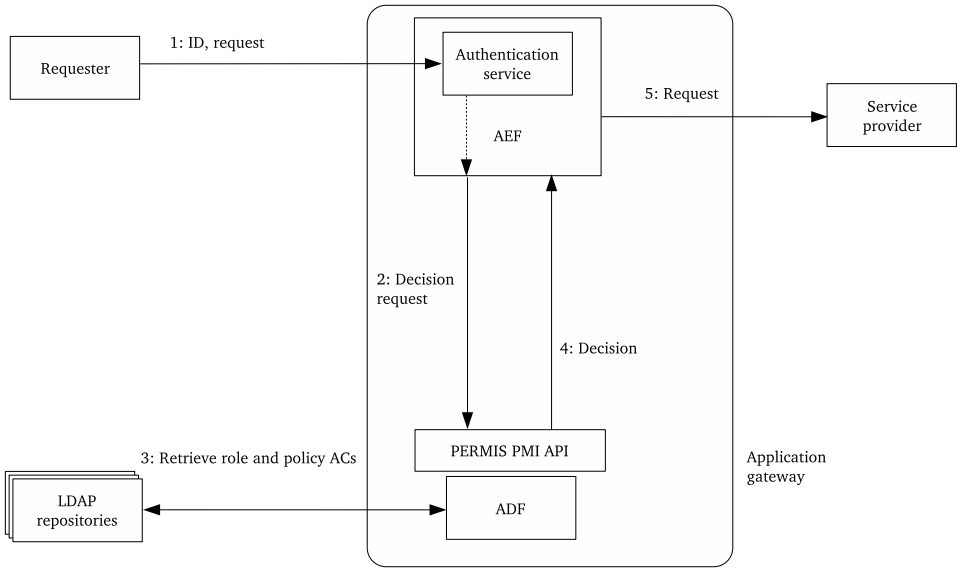

Figure 2.12: The privilege verification subsystem of PERMIS.

The main assumption of PERMIS is that the set of users is known in advance and that a security administrator has assigned these users to predefined roles. However, in pervasive computing the complete set of participating users cannot be known in advance since as

we have seen such environments are volatile and highly dynamic. Moreover, PERMIS relies on an infrastructure of LDAP repositories for the issued role and policy certificates, an assumption that is not compatible with the vision of pervasive computing. Finally, PERMIS provides no support for passing values of dynamic variables from the service provider to the ADF in order to facilitate context-aware authorizations. However, partial context-awareness can be accomplished by encoding contextual information in policy ACs.

2.6.9 XACML

The eXtensible Access Control Markup Language (XACML) [OAS05] is an XML-based generic and flexible language to model access control policies. The OASIS XACML standardization committee provides XML schemas to allow the expression of authorization policies in XML for protecting resources that are also expressed in XML. The main component of an XACML policy is a *rule*. A policy can have more than one rule and each rule consists of three sub-components; a target, an effect, and a condition. The target sub-component specifies the set of resources, subjects and actions to which the current rule applies. The effect can be either permit or deny. The condition sub-component defines a boolean expression and is used to allow the specification of conditions that must be satisfied in order for the current rule to apply. An XACML request is an XML document describing the subject, the attributes associated with the subject, the target resource and the actions that the subject requests to be performed on the target resource. An XACML response can be either permit, deny, not applicable that is returned when no matching rule is found for the given request, or indeterminate that is returned when an error has occurred.

Contextual information can be used to define XACML conditions, however the language provides no support to allow the definition of variables with dynamic values. This limits the ability to have context-aware policies that are able to adapt to the current context of the service provider. Another limitation of XACML, that indeed affects all XML-based systems, is its poor performance when compared to binary or even plaintext alternatives [MC04]. Finally, the delegation model supported by the 2.0 version of the XACML standard is simple and restrictive. A user is able to express that another user is trusted to

perform a certain action on his behalf, but there is no way to create policies that allow the dynamic creation of further policies.

2.7 Pervasive Computing Security Systems

In this section we examine several pervasive computing security systems and the way they address the problem of authorization. Moreover, we survey security systems that have been proposed for ad hoc networks since we feel that several of the solutions try to handle similar problem areas. We analyze their operational requirements focusing on whether they manage to satisfy the three previously identified key properties of pervasive computing; open and dynamic nature, graceful and unobtrusive usage models, and context-awareness.

2.7.1 Resurrecting Duckling

The Resurrecting Duckling security model was proposed to cover master-slave type of relationships between two devices in ubiquitous computing environments [SA00]. The relationship is established when the master device exchanges over an out-of-band secure channel a secret piece of information with the slave device. Stajano and Anderson call this procedure *imprinting*. The imprinted device can authenticate the master through the common secret, which is a symmetric cryptographic key, and accept service requests from it. The basic idea behind the Resurrecting Duckling is that a dual interface is used. One interface is used to implement the out-of-band secure channel and to bootstrap the security association between the master and the slave devices. Stajano and Anderson propose the use of physical contact between two surfaces of the devices as an example of such a channel. The other interface is a wireless network interface and is used to actually perform the service access request and reply protocols between the two devices. The protocols that utilize the wireless interface are secured using the shared secret key exchanged over the out-of-band secure interface.

The main advantage of the Resurrecting Duckling model is that it is able to operate and establish secure associations between devices in a disconnected manner, without the need to contact a TTP or to have access to an online PKI. Although human intervention is still required to establish the secure out-of-band channel, this can be naturally inte-

grated to the physical tasks the operators of the devices perform to achieve their goals. However, its usefulness is limited in facilitating exchanges between peer devices since the master-slave communication model is not able to address all types of interactions used in pervasive computing. Hence the basic model was extended to allow the imprinting of devices with policies that define the types of relationships a slave device is allowed by its master to have with others in order to address peer-to-peer interactions [Sta01]. This extended model basically allows a master to delegate the capability of using its slave devices to other participants of the environment. The embedded policy can be arbitrarily complex and specifies the credentials that are required to allow the execution of a specific operation. The slave device based on the credentials exhibited by the requesting peer and the embedded policy allows or not the execution of the requested operation. However, the authors simply proposed the use of trust management systems as a way to define the imprinted policies and the credentials without offering any specific engineering details on how the authorization process works. For example, the model does not define if credentials issued by principals other than the master are acceptable in certain scenarios. By allowing only masters to issue valid certificates even the extended Resurrecting Duckling system defines a static association model between the master and the imprinted devices, limiting its direct application in situations where associations are established in an ad hoc manner. Furthermore, the problem of revocation of previously issued credentials or policies is not addressed by the model. Finally, there is no support for using external information from the environment in order to facilitate context-sensitive authorizations.

2.7.2 Talking to Strangers

The idea of bootstrapping trust relationships in adversarial environments over secure out-of-band channels was examined further by Balfanz et al. [BS02] and their concept of *location-limited* channels. These are channels that have the property that human operators can precisely control which devices are communicating with each other thus ensuring the authenticity requirement. They propose the use of contact, infrared and audio as a way of exchanging pre-authentication information without requiring secrecy from the out-of-band channel. They employ the use of public key cryptography and they exchange

digests of the public keys of the peers as the pre-authentication information over the established location-limited channel. Therefore, each peer is sure about the authenticity of the key the other peer presents in the following steps of the protocol. This basically removes the requirement of a PKI with the requirement of close physical proximity between the two devices that need to communicate securely, which is more appropriate for pervasive computing. Another advantage of having location-limited channels that are not secret is the ability of establishing secure group communications. Certain location-limited channels are broadcast in nature, such as audio. Consequently, non-secret pre-authentication information can be transmitted over them and be utilized to bootstrap secure group communication protocols. Although the Talking to Strangers (TTS) model provides a more flexible definition of out-of-band secure channels, it generally suffers from the same problems as the Resurrecting Duckling model. It does not address the authorization process, it avoids to address revocation, and does not include a way to use context-awareness in the establishment of security associations.

2.7.3 Bluetooth Security

Bluetooth is a technology used for short range wireless communications. It has been developed to address the local connectivity needs of mainly portable and handheld devices, like mobile phones, PDAs, digital cameras, pagers and others. In contrast to infrared, Bluetooth does not require line-of-sight between the peer devices, which form *piconets* to communicate. Each piconet comprises of up to eight active Bluetooth devices where one is the *master* and the rest are *slaves* [Blu03]. The master device is responsible to admit slave devices to the piconet and assign them Bluetooth addresses. Each piconet may have only one master, but one master may be a slave device in another piconet, creating *scatternets*. Since a Bluetooth signal cannot be confined inside a certain physical area, it is possible for an attacker to eavesdrop on the data channel and capture the exchanged information. Therefore, Bluetooth has a significant security component that offers authentication, confidentiality and to a very limited extend authorization for the different services provided by the devices. The two main procedures of Bluetooth security are *pairing* and *authentication*. Pairing is used to authenticate two peer devices and create a

common link key for establishing a security association. Before the initiation of the pairing procedure the same PIN must be entered into both devices. Furthermore, each device has a 48-bit unique address, known as the Bluetooth Device Address (BD_ADDR). The master device generates a random 128-bit number (R_0) and transmits this in plaintext to the slave device. Then both devices use the E_{22} algorithm to derive K_{init}, the initialization key which is 128 bits long and is used for a single session between devices that have not communicated before. Based on K_{init} the devices create the link key K_{link} with the following procedure. Each device generates a new random number (R_M the master and R_S the slave), performs a bitwise XOR operation on it using K_{init} and sends it to the other device. Based on the previously agreed K_{init}, both devices now have both random numbers. These random numbers are then used as input to the E_{21} algorithm to derive K_{link}. The whole pairing procedure is illustrated in Fig. 2.13.

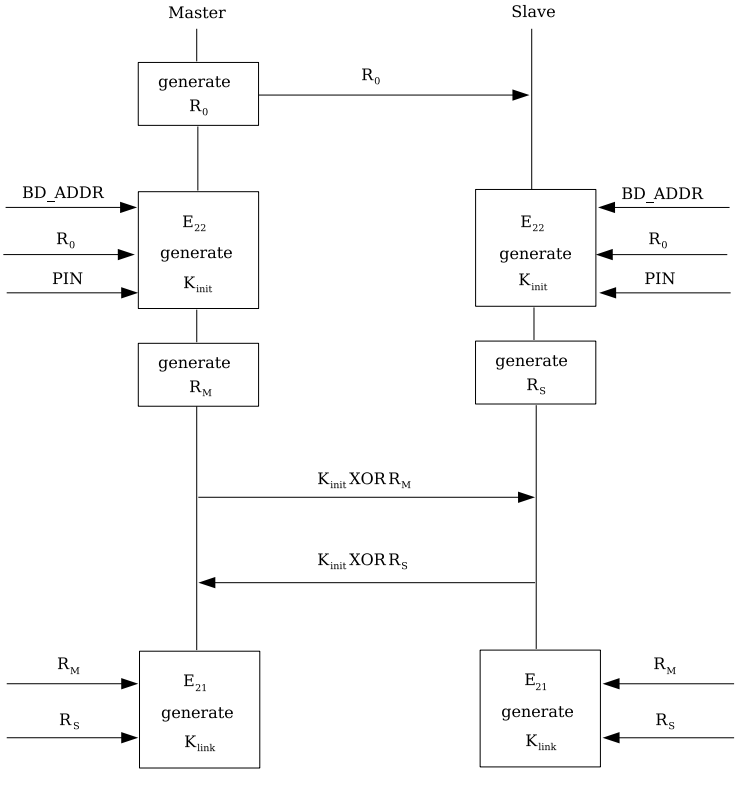

Figure 2.13: Bluetooth pairing process.

The goal of the authentication procedure is to verify the identity of a remote Bluetooth

device. It is based on a challenge-response protocol that uses a randomly generated number by the verifier and the previously agreed K_{link} link key. The verifier generates an 128-bit random number (R_V) and sends it in plaintext to the claimant. The claimant receives R_V and by using this and K_{link} as input to the E_1 algorithm gets a 32-bit number (R_C). This is send to the verifier who compares it to the result of its own E_1 application and if the comparison is successful the verifier and the claimant change roles and perform the same process again in order to achieve mutual authentication. Fig. 2.14 presents the authentication procedure.

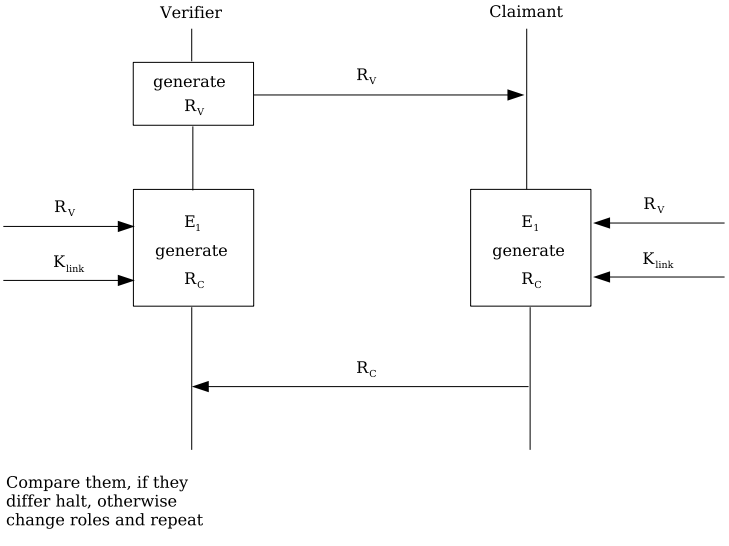

Figure 2.14: Bluetooth authentication process.

The Bluetooth security component also maintains trust levels for the different devices the current device has been associated with. These levels are defined based on whether an authentication procedure has taken place between the devices, and on whether the user has specified a peer device as "trusted" (see Table 2.3).

The device trust classification allows a provider to determine whether to allow a requester to access a provided service. Each service is also assigned by the user to a class. These classes are the following three:

1. Services that require both authentication and authorization.

2. Services that require only authentication.

Table 2.3: Bluetooth trust levels.

Trust level	Meaning
Trusted	A device that has been previously authenticated, a link key has been established and the user has marked it as "trusted" in the device database. Such devices have unrestricted access to all provided services.
Untrusted	A device that has been previously authenticated, a link key has been established but the user has not marked it as "trusted" in the device database.
Unknown	No information is available for this device.

3. Services that are open to all devices.

The authorization support provided by the Bluetooth security component is very limited. If a service is classified as one that requires authorization, the device requests this explicitly from the user. The devices that are defined by the user as "trusted" are always allowed access automatically. However, all the other peer devices require manual authorization. We believe that this can become obtrusive if there are a lot of devices in an environment, or if there is a great number of provided services. As an example of the explicit manual Bluetooth authorization we include here the relevant dialog from a Windows XP device providing a service; the device named "Test" is the one requesting a service and it has not been labelled as "trusted" by the user of the Windows XP device (see Fig. 2.15).

Figure 2.15: Bluetooth explicit service authorization request.

The Bluetooth security architecture removes the requirement of having an online centralized TTP by relying on PIN codes that must be entered into the two devices that need

to communicate. Also, incoming access requests by untrusted devices must be manually authorized by the user of the providing device. Although these two requirements may not seem to be particularly obtrusive in environments with few devices, as the number of participating devices and services grows it becomes a constant nuisance to users. This leads to most users simply labelling all devices they come across as fully trusted without considering the security repercussions. Furthermore, the Bluetooth standard includes no support for allowing context data to influence authorization decisions. Finally, Bluetooth security has been heavily criticized and a lot of cryptanalytical [FL01], [HN99], [LV04] and brute force [SW05] attacks have appeared in the literature during the last years.

2.7.4 Distributed Key Management

A solution to the problem of having a single TTP in a dynamic networking environment certifying the public keys of participating entities was presented by Zhou and Haas [ZH99]. Their approach aims at addressing the problem of key management in ad hoc networks. Instead of relying on a single CA or a replicated CA at different nodes, which aggravates the single point of failure/attack problem, the authors proposed the use of *threshold cryptography* to divide the private key of the CA service between n arbitrarily chosen nodes of the ad hoc network. Any $t+1$ of these n nodes can jointly perform a signature generation operation in order to produce a public key certificate for a new node. The system is able to tolerate up to t compromised CA nodes since $t+1$ partial signatures are required in order to produce a full valid signature. *Share refreshing* techniques that periodically create new sets of private key shares are also used. Thus, a mobile attacker has to compromise at least $t+1$ CA nodes within the time period of two consecutive share refreshing processes in order to compromise the private key. Although this approach facilitates more flexible key management than traditional PKI systems, it assumes that somehow certain nodes are chosen to serve the special purpose of CA share nodes. This assumption cannot always be realistically satisfied in all ad hoc networking or pervasive computing applications. For example, in application scenarios that involve everyday and not military or emergency activities the selection of the CA share entities becomes an obstacle.

A similar solution was proposed in [KZL+01] by Kong et al. Again the private key of

the CA service is divided into a number of shares. The main difference with the previous scheme is that *any* participant of the ad hoc network can have a share of the private key. A node is given a public key certificate binding its node identifier to its public key by using $t+1$ partial signatures from the nodes that hold shares of the private key. Although this scheme still requires an authority to prime the initial $t+1$ nodes with private key shares, it copes better than the previous scheme with high mobility scenarios since any node can be a potential CA share node. The shares are rearranged whenever a new node that wants to acquire a share of the private key obtains one by the $t+1$ nodes that already have shares. However, the scheme can be attacked by a malicious node that repeatedly changes its identifier and acquires all the necessary number of shares to reconstruct the private key of the CA service.

Another problem with both systems lies in the fact that the devices that perform the CA share node role are assumed to always be available. At anytime a partial signature request may be addressed to them. Therefore, they have to constantly be in the vicinity of the local networking environment in order to be able to respond. The authors do not specify if the share nodes have the ability to delegate their role to other participants of the network. Finally, even with the central CA entity requirement removed, identity-based security infrastructures offer little help in addressing the authorization requirements of the volatile pervasive computing paradigm.

2.7.5 Self-organized Public Key Infrastructure

A PKI-based key management approach for mobile ad hoc networks has been proposed by Capkun et al. [CBH03]. Their proposed solution avoids completely a universally trusted third party, or a CA, by using self-signed certificates in a manner similar to the PGP web of trust. Based on personal real-world trust considerations the users issue public key certificates to each other. All users have to maintain a personal repository of certificates that they have issued to others and of certificates that others have issued to them. Certificate revocation is handled *explicitly*, the user informs communicating peers about the status of revoked certificates, or *implicitly* by using short expiration times and renewal of previously issued certificates. The goal of the scheme is to enable a source node that wishes to

communicate with another node to obtain the destination's authentic public key. As an example consider the case where source node A wishes to communicate with destination node B. Nodes A and B merge their certificate repositories and A tries to find a public key certificate chain that connects it to B. The chain has to be constructed in a way that the first certificate can be verified using A's public key, and each next certificate can be verified with the key included in the previous certificate of the chain. The last certificate of the chain must include the public key of node B. The main problem with this approach is that users are assumed to issue valid certificates. To deal with malicious participants issuing false certificates, the authors introduce confidence metrics to measure the extent that a certificate can be trusted. Furthermore, the authors assume that trust between participating entities is transitive, that is if A trusts B and B trusts C then necessarily follows that A must trust C, which is not always a valid assumption [CH96].

2.7.6 Ad hoc Group Key Agreement

Another key management approach designed to address scenarios of room meetings has been proposed by Asokan and Ginzboorg [AG00]. As users come into the meeting room they are given, or read from a blackboard as the authors suggest, a commonly shared password. To enhance the security of the scheme against brute force attacks, the password is not used directly, but a strong shared key is derived from it. The shared key derived from the password can either be the same for every participant (*group key exchange*) or a different one for every exchange between two participants (*two-party key exchange*). Since this scheme has been designed to address a specific application scenario it cannot be easily applied to different situations. In pervasive computing where mobility is high and membership changes frequent this key management approach becomes cumbersome.

2.7.7 Proxy-based Security Protocols

Based on the assumption that in pervasive computing most devices will have limited processing and memory resources, Burnside et al. proposed the use of virtual proxies employed to offload tedious calculations [BCM+02]. While personal user devices are assumed by the authors to be computationally restricted, these infrastructure virtual proxies have high-

bandwidth connections and copious resources available. Their proposal presents a security architecture based on SPKI/SDSI that offers authentication, authorization and confidentiality services to proxy-based pervasive computing solutions. Every device in the system, like household appliances, mobile phones, handheld computers, and others, have associated trusted software proxies that run on infrastructure computers. Two security protocols are analyzed, one that secures the device-to-proxy communications, and one that secures the proxy-to-proxy communications (see Fig. 2.16).

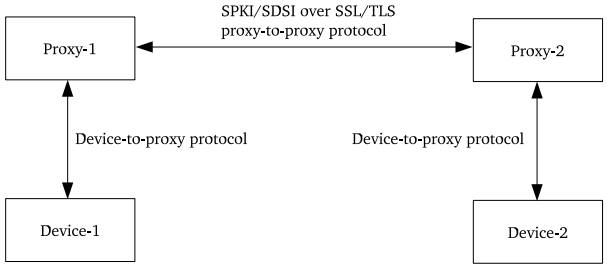

Figure 2.16: Proxy-based security protocols.

The device-to-proxy protocol is described by the authors very briefly; every device and its associated proxy are assumed to share 128-bit keys. These shared keys are then used to encrypt and authenticate communications between the two components. However, there is no explanation of the mechanism used to initialize these keys. The virtual proxy is responsible to make access control decisions on behalf of the device it represents. The authorization process is based on SPKI/SDSI capability certificates; the requester presents the access request to the proxy which responds with the relevant ACL. Then the requester sends the SPKI/SDSI certificates she holds to support the request and satisfy the ACL. If the proxy correctly verifies the certificates and discovers that the request is authorized it forwards the requested action to the associated device over the device-to-proxy protocol. Furthermore, the authors propose the layering of the proxy-to-proxy protocol on top of SSL/TLS in order to provide further security services, like confidentiality for the exchanged certificates and ACLs. However, it is not clear how this is accomplished. SSL/TLS uses the X.509 PKI to authenticate peers based on PKCs binding their identifiers (usually DNS names) to their public keys. We see two problems with this approach. First, if the X.509

PKI is required then the proposed solution becomes inappropriate to address the security concerns of pervasive computing due to its centralized nature and inability to operate in a disconnected manner[10]. Second, if self-signed X.509 PKCs are used then it is possible for an attacker to perform a man-in-the-middle attack against the authentication step of the proxy-to-proxy protocol.

The main shortcoming of the proxy-based security protocols is that they require all communication traffic between pervasive devices to flow through their associated proxies. These proxies require fixed infrastructure and manual configuration. In addition, the proxy infrastructure needs to be persistent and always accessible, otherwise no secure associations can be established. This comes in contrast to the decentralized and disconnected paradigm advocated by pervasive computing which requires users to be able to communicate even when they cannot access infrastructure networks.

2.7.8 Personal PKI

The term "personal PKI" describes a PKI-based security solution specifically designed to support the distribution and management of public keys and PKCs in a Personal Area Network (PAN) [Mit04]. The authors define a PAN as a collection of portable or moving devices in close proximity to a physical person. The main assumptions of the system are threefold. First, that a PAN contains at least one device performing the role of the PAN's root CA, called the personal CA. Second, that devices of the PAN cannot rely on either existing symmetric shared keys or connectivity to a TTP in order to establish secure associations between themselves. Third, that the personal CA and the other participating devices must be equipped with simple input and output devices. Based on these assumptions a *device initialization* protocol is described in which the personal CA securely transfers its public key to a device of the PAN and issues a PKC for it. The communication between the two devices takes place over an insecure wireless interface and since all information to be transfered is assumed to be public the protocol protects only the integrity of the exchanged data and not their confidentiality [Mit04]. This is accomplished by utilizing the input/output facilities of the devices and by requiring from the user to read the digest

[10] For more details on why the X.509 PKI is not compatible with pervasive computing please see subsection 2.6.2.

of a randomly generated key from one device and enter it into the other, thus ensuring integrity (see [Mit04] for the complete protocol).

The authors also analyze the applicability of their protocol in the cases that one or both devices lack keypads or displays. Moreover, revocation issues in the personal PKI are examined, and the use of Current Identity Lists (CILs) is proposed. CILs are lists that include the valid PKCs, which is basically the opposite concept of CRLs. CILs are applicable to the personal PKI scenario since the number of participating entities is typically small. Although the concept of the personal PKI seems to fit nicely with the decentralized security requirements of pervasive computing, it is still an identity-based authentication architecture; the authorization problem must be addressed separately and all attempts to solve it present the same difficulties as with other PKI-based solutions.

2.7.9 Cerberus

The Gaia project aims to define a generic computational environment that integrates physical spaces and the pervasive computing devices that exist in them into a programmable system [RHC+02]. Cerberus is a core service in the Gaia platform that provides identification, authentication, context-awareness, and reasoning [AMRCM03]. It utilizes an infrastructure of sensors that are installed throughout the environment in order to capture and store as much context information as possible. This information is then used to identify entities and reach decisions regarding the actions they request to perform. There are four main components that collectively provide this functionality:

1. The security service which is responsible for identifying and authenticating entities. Each authentication method is associated with a *confidence value* that represents how confident the system is about the identity of an entity. Each confidence value is a number in the range [0, 1]. Entities can use multiple authentication methods, like smart badges or fingerprint scanners, to increase their confidence value. This metric is then used in access control decisions. The security service also provides an authorization API which applications can use to check if certain entities can perform particular operations.

2. The context infrastructure that uses first-order predicate calculus and boolean alge-

bra to allow the definition of complex access control rules involving context. This is in essence a monolithic context database that stores all possible current context information the sensors of the system can collect. Furthermore, context history is also maintained in the database allowing a persistent view of events taking place in the environment.

3. The knowledge base which contains access control policies also expressed with first-order logic. Cerberus has two different kinds of policies; the ones that determine the confidence value of an entity based on the authentication method it uses, and the ones that define authorizations. The latter use the confidence value associated with an entity to decide if access to a particular resource is allowed or denied.

4. The fourth component is the inference engine which is responsible for performing two tasks. The first one is to assign confidence values to entities based on the authentication policies and the contextual information stored in the context infrastructure database. The second one is to evaluate access queries from applications running in the environment regarding whether an entity can perform a certain operation on a resource. Decisions are reached based on the access control policies, the credentials the entity presents, its associated confidence value, and its current context data.

Fig. 2.17 presents the Cerberus system and its four main components. The main disadvantage of Cerberus lies in its monolithic architecture and its reliance on a central server to collect context information, evaluate policies and make authorization decisions on behalf of resource providers. This implies the requirement that all the resource providers of the environment trust the server to give them correct authorization decisions and not to reveal their personal data, which may be private in nature. However, pervasive computing entities (both resource providers and consumers) establish short-lived associations and frequently roam between administrative domains and hence do not necessarily trust each other. Additionally, the central component that reaches authorization decisions needs to be constantly accessible and therefore forbids disconnected operation.

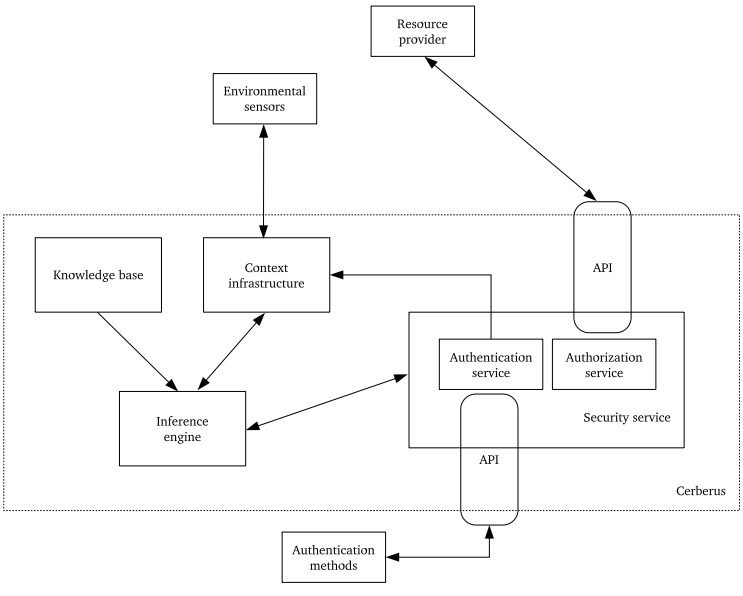

Figure 2.17: The Cerberus system.

2.7.10 Environment Roles

Covington et al. proposed the concept of *environment roles* in order to develop an access control system for pervasive computing that can utilize environmental and contextual information [CLS+01]. Environment roles are in essence one component of the Generalized Role-Based Access Control (GRBAC) model [CMA00]. GRBAC is an extension of the traditional RBAC model where object and environment roles are defined in addition to subject roles. The resources provided in an environment can be assigned to object roles and environmental (contextual) conditions are used to define environment roles. GRBAC uses a logic language similar to Prolog to express access control constraints on context variables. Authorizations are based both on traditional subject roles and environment roles. This allows the system to be sensitive to contextual information, which is captured by sensors and translated to specific environment conditions. These conditions are then used to activate environment roles for the subjects in the smart space based on logic predicates.

GRBAC constitutes a flexible set of extensions to RBAC; the concept of object roles greatly simplifies the administration of resources and the definition of access control policies. Based on captured context, environment roles can be activated automatically for

subjects in the smart space giving them certain access rights without the need of manual configuration tasks. However, GRBAC relies on a central management service. This service has two responsibilities. The first is to collect context data and maintain environment conditions. The second is to store the logic predicates that implement environment role activation decisions and authorizations. All resource consumers and providers that participate in the system need to have persistent access to the management service, otherwise they cannot activate environment roles or reach authorization decisions.

2.8 Secure Service Discovery Systems

This section presents a brief examination of secure service discovery systems. Although this is not our main research area, we believe that the problems of authorization and secure service discovery are interrelated, especially in the domain of pervasive computing. As we have already discussed in a previous section, service discovery is an enabling technology for pervasive computing as it allows users to locate the services provided in an environment without requiring tedious administrative and configuration tasks. Services need to be discovered before an access request can issued. However, the fact that a service can be discovered by an entity does not necessarily imply that it can also be accessed by the same entity. In addition, as we examine in greater detail in the following chapter, unauthorized access to service advertisements leads to security threats.

2.8.1 Jini

The Jini [ASW+99] service discovery framework utilizes a component called the Jini Lookup Service (JLS) to allow entities in an environment to discover provided services. When an entity wants to provide a new service it locates the JLS of the local network and sends to it a Java object that implements the access interface of the service. The interface also serves as a template for client search requests. We believe that the introduction of a central component as the JLS in a pervasive computing environment introduces additional administrative overhead.

Jini provides very limited support for satisfying security requirements, as this has not been one of its design goals. Authorization services are offered only when a client tries to

access a service, but not during the process of discovery. Therefore, any client is able to discover all the services of an environment, even when it is not allowed to access any of them. Additionally, the JLS does not support any kind of authentication; unauthenticated services may register, unauthenticated clients can attempt to access registered services, and rogue JLSs can be set up.

2.8.2 Service Location Protocol

The Service Location Protocol (SLP) [GPVD99] has three main components; the User Agent (UA), the Service Agent (SA) and the Directory Agent (DA). The UA runs on client entities interacting with the user applications and sending service discovery requests. The SA is responsible to generate and dispatch advertisements for the services that an entity wishes to provide. The DA implements a central service information repository that receives and caches the advertisements sent by the SAs of the local network and replies to requests from UAs. SLP is also able to function in a fully decentralized manner without the presence of DAs; UAs multicast their service requests and the relevant SAs unicast their replies directly to them.

SLP provides limited authentication by allowing service request messages to carry information that protect the integrity of the message. However, it does not include any mechanism to permit SAs or DAs to authenticate UAs. Finally, SLP does not support any kind of access control for the provided services and leaves this to be addressed by each SA independently.

2.8.3 Universal Plug and Play

Universal Plug and Play (UPnP) [UPn03] uses the Simple Service Discovery Protocol (SSDP) for service discovery and is able to operate both with and without a central service registry in the local network. The former is accomplished by having service providers advertise their services using multicast messages, which include the services' types, names and URLs that point to XML files with the access interfaces. UPnP control points in the network register the advertisements and unicast them to the appropriate requesting clients. In the decentralized mode of SSDP clients broadcast their requests in the network

and the matching service providers reply with unicast messages containing URLs to XML files describing the service's access interface.

The process of service discovery is protected in SSDP by announcing all devices as generic devices without any details regarding the services they provide, except in response to access requests from authorized clients. Service access control is provided by having an ACL in each device that lists the unique IDs of client devices and the operations they are allowed to perform on the current device.

2.8.4 Secure Ad hoc Service Discovery

Zhu et al. proposed a proxy-based secure protocol for discovering and accessing unfamiliar services in public ad hoc networking environments [ZMN03]. Their model of establishing secure communications utilizes channels outside the local network to aid pervasive computing entities to authenticate and authorize each other. Based on this assumption they have defined two models, one that does not use third-generation mobile telephone technology (3G) channels and one that does.

In the first model of accessing an unfamiliar service without a 3G connection, the client device must access its proxy through the Internet connection the service provider is assumed to have. The service provider authenticates itself to the client's proxy with the help of an assumed existing PKI. Then the proxy generates a session key and sends it to both the service provider and the client (through the provider). The service provider then decides whether to allow access to the client based on the identity of its authenticated proxy. In the second model, when a client wishes to access a service it sends the service request and its certificates directly to its associated proxy over a 3G channel. The proxy then contacts the service provider over the Internet and requests access. If the provider authorizes the request it sends a generated session key to the proxy which is then forwarded over the 3G connection to the client.

Both models rely on a centralized PKI-based trust model. Furthermore, the local network is assumed to always have a persistent connection to the Internet in order to contact both the PKI system and the requesting client's proxy. Finally, neither of the proposed protocols addresses the protection of service advertisements.

2.9 System Classification

Table 2.4 classifies the security systems, both the pervasive and the traditional ones, we have surveyed according to the previously identified requirements. Table 2.5 classifies the service discovery systems we have examined. Our classification demonstrates that no single existing system satisfies the complete set of requirements for authorization management in pervasive computing environments.

Table 2.4: System classification according to requirements.

Systems	Authorization support	Decentralized management	Disconnected operation	Context-awareness	Unobtrusive usage model
Ferraiolo et al. [FK92]	Yes	No	No	No	No
ISO/ITU-T [ISO01]	Yes	Yes	No	No	No
Farrell et al. [FH02]	Yes	No	No	No	No
Zimmermann [Zim95]	No	Yes	No	No	No
Kohl et al. [KN93]	Partial	No	No	No	No
Blaze et al. [BFK99]	Yes	Yes	Yes	No	No
Rivest et al. [RL96]	Yes	Yes	Yes	No	No
Herzberg et al. [HMM$^+$00]	Yes	Yes	No	No	No
Chadwick et al. [CO02]	Yes	Yes	No	Partial	No
OASIS XACML [OAS05]	Yes	Yes	No	Partial	No
Stajano et al. [SA00]	No	Yes	Yes	No	Yes
Balfanz et al. [BS02]	No	Yes	Yes	No	Yes
Bluetooth SIG [Blu03]	Partial	Yes	Yes	No	No
Zhou et al. [ZH99]	No	Yes	Yes	No	No
Capkun et al. [CBH03]	No	Yes	Yes	No	No
Asokan et al. [AG00]	No	Yes	Yes	No	No
Burnside et al. [BCM$^+$02]	Yes	Yes	No	No	Yes
Mitchell et al. [Mit04]	No	Yes	Yes	No	Yes
Al-Muhtadi et al. [AMRCM03]	Yes	No	No	Yes	Yes
Covington et al. [CLS$^+$01]	Yes	No	No	Yes	Yes

Table 2.5: Service discovery system classification.

Requirements	Arnold et al. [ASW+99]	Guttman et al. [GPVD99]	UPnP Forum [UPn03]	Zhu et al. [ZMN03]
Authentication support	No	Partial	Yes	Yes
Authorization support	Partial	No	Yes	No
Decentralized management	No	Yes	Yes	Yes
Disconnected operation	Yes	Yes	Yes	No
Service discovery protection	No	No	Partial	No

Chapter 3

Pervasive Computing Threat Model

The survey of the areas presented in the previous chapter directly leads to the formulation of a comprehensive threat model that helps in identifying what we are trying to protect against when we design a security system for pervasive computing. The definition of a threat model is an essential part of the design of any security system. When a security model is designed in order to protect a specific application against attackers certain assumptions are made. These assumptions are necessary in order to make the end product usable by legitimate users. However, a completely trusted environment cannot be realistically expected; attackers will discover the assumptions made by the system and will try to take advantage of them to increase their authority. Therefore, it is very important to understand the possible threats, identify the avenues of attack and enumerate all the assumptions made by the system we are trying to secure. Threat models do not attempt to define the constraints that a system places on the attacker to satisfy its security properties. This is accomplished in the complete security model of the system under development. Hence, a threat model is a subset of a system's security model, and the first step towards the realization of the latter. Although the term "pervasive computing" refers to an entire computing paradigm and not a specific user application, we believe that a threat model is necessary in order to help us develop a security model than can focus on protecting against certain threats and manage the related assumptions. One methodology to discover and list all possible security attacks against a system is known as *attack trees*. To create an attack tree we represent attacks against a system in a tree structure; the attack goals as

root nodes and the different subgoals necessary to achieve them as their leaf nodes [Sch99].

Previous efforts to define a threat model for pervasive computing have been less than complete. Projects published in the literature on the area of secure pervasive computing present a limited view of threats and are focused on what the proposed system aims to address. For example, the Talking to Strangers (TTS) system [BS02] that we have already examined in the previous chapter explores only the problem of authenticating devices in order to avoid sending confidential data to other devices than the one the user intended to. The solution proposed by the TTS system is based on Stajano's Resurrecting Duckling model [SA00] and uses location-limited channels to perform authentication between devices. The presented threat model focuses only on attacks against the establishment of such a channel. Asokan and Ginzboorg also defined a threat model specific to their group key protocol. They generally assume a *strong* and a *weak* attacker; the strong attacker is able to disrupt any protocol by modifying the communication channel among the legitimate participants, and the weak attacker can only insert messages but cannot modify or delete messages sent by others [AG00]. Burnside et al. in [BCM+02] give a very limited threat model for their proxy-based authentication protocol. Their main focus is replay attacks and the use of timestamps to protect against them. A more complete representation of threats was given by Keely [Kee01]. Although he focuses on threats to wireless mobile networking for e-business applications, several of the attacks he identified are applicable to pervasive computing environments and applications. Creese et al. [CGRZ03] focus on formalizing the ubiquitous computing threat model examining mainly the assumption of a dual interface; one interface for establishing location-limited channels and bootstrapping security associations, and one for actually participating in networking protocols. Although context-awareness is an inherent characteristic of pervasive computing and one that opens new avenues of attack, none of the surveyed projects addresses it in their threat model.

Fig. 3.1 illustrates the pervasive computing threat model we have defined using the attack trees methodology. During the development of the model we have identified that several attacks lead to other attacks which we have previously included and analyzed. These are represented in the tree as identical nodes in different locations. A node that appears in more than one location of the tree has the same sub-tree everywhere. In order

to avoid including the same sub-tree multiple times in the model we have used icons to denote duplicate nodes. Therefore, a node marked with an icon means that it exists somewhere else in the tree and if that node has a sub-tree this sub-tree is included only once. Furthermore, the textual description of some nodes is highlighted. These nodes do not represent attacks but reasons for why solutions based on the parent nodes are not suitable for pervasive computing systems. We are using such an extension to the attack trees methodology in order to have a complete overview of pervasive computing security in our model.

We have identified seven different general categories of attack on a pervasive computing system; Context-sensitivity, unauthorized actions, eavesdropping on the communication channel, threats associated with service and device discovery, denial of service (DOS) attacks, stolen devices, and attacks focusing on a secure transport layer protocol solution (like for example the Transport Layer Security (TLS) protocol). Although we have not explicitly included privacy threats in our model, a lot of the enumerated attacks result in reducing the privacy of pervasive computing users. Moreover, our model does not include the social aspects of pervasive computing security since we focus on technology-related attacks. We do believe that in future computing environments social-based attacks such as *social engineering* [MSW03] will be even more effective than in traditional computing, but we consider these to be outside the scope of our model. In the following sections we will analyze the attack categories in detail and discuss the possible privacy implications they introduce.

3.1 Context-sensitivity Threats

Pervasive computing systems rely on context-awareness in order to both minimize manual intervention from the users for possible required configuration tasks and to enhance their understanding of the surrounding environment. This reliance opens new avenues of attack on pervasive computing systems and applications that do not exist in traditional computing. A close examination of the enumeration-based context definition given by Adelstein et al. [AGRS05] reveals that we can sort the different context categories based on the novelty of possible attack avenues their use opens (see Fig. 3.2).

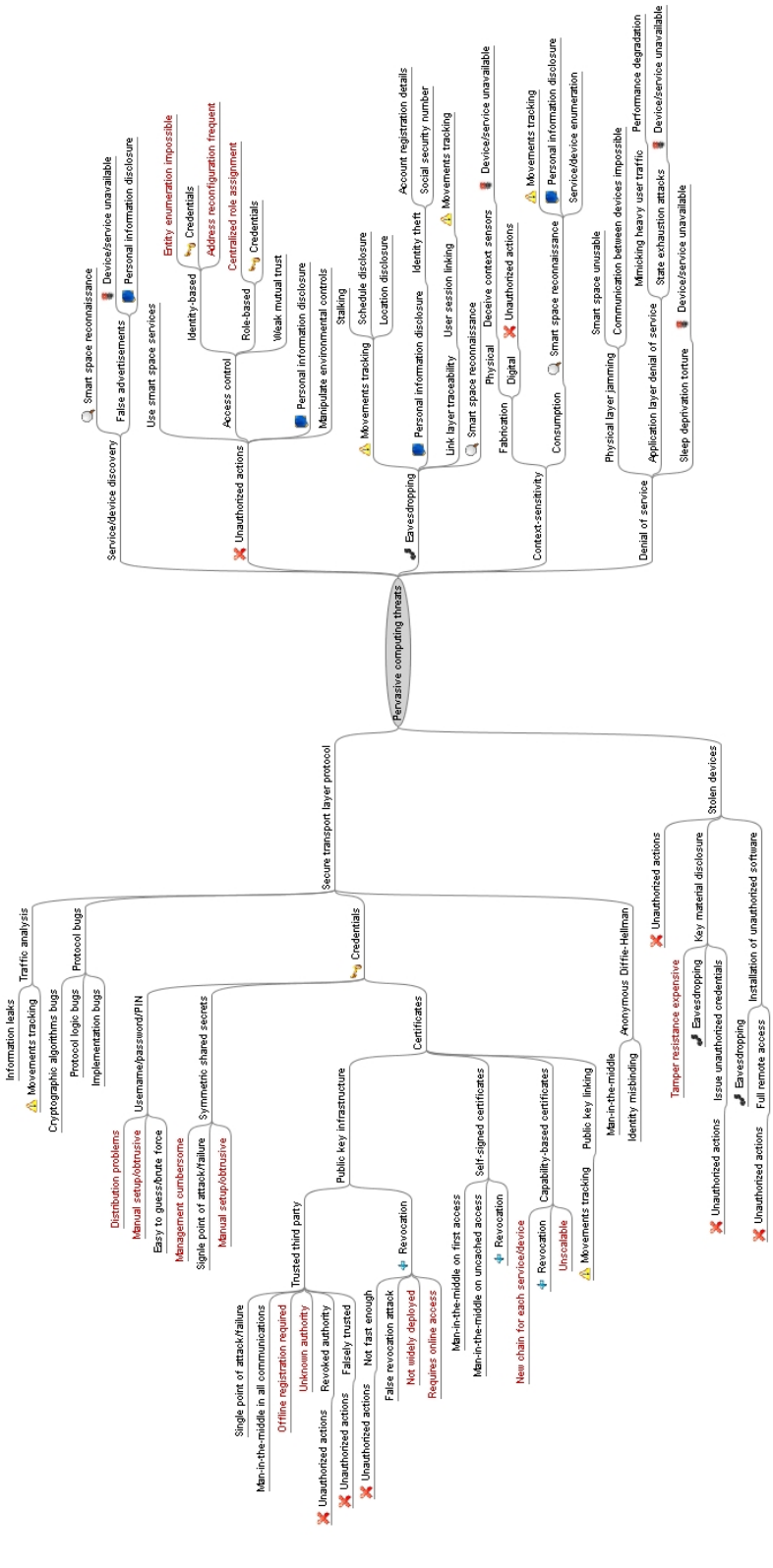

Figure 3.1: Pervasive computing threat model.

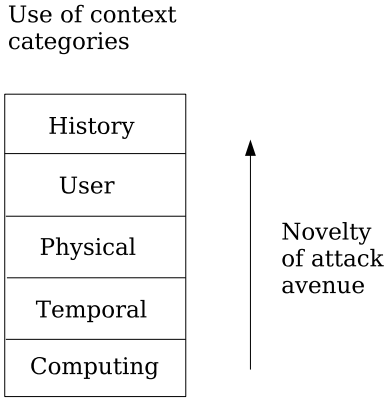

Figure 3.2: Use of context categories and novelty of attack avenues.

As context-sensitive applications utilize more sophisticated context categories to benefit their human users the possibility of creating new attack patterns and vulnerabilities that did not exist before increases. An example considers the computing context category, which includes information regarding network connectivity, communication costs, available bandwidth, nearby accessible resources and others. An attacker that has access to such kind of data can attack the system only in ways that have been documented and are known in conventional computing. Known security vulnerabilities of specific versions of network servers, unpatched operating system services and cleartext application protocols all fall into this category. However, as we move upwards to the use of more complex context categories the attack avenues become more novel and undocumented thus far. This section identifies these new forms of attacks.

3.1.1 Context Fabrication Threats

We define context fabrication threats as those in which a malicious entity creates context information that does not correspond to any physical or digital environmental parameters. The relevant section of the threat model is illustrated in Fig. 3.3. The fabrication of non-existent physical parameters can directly lead to providing false data to the sensors of a pervasive computing system. Consequently the system may deny access to a legitimate user since it has taken into consideration the false physical context data. For example, we

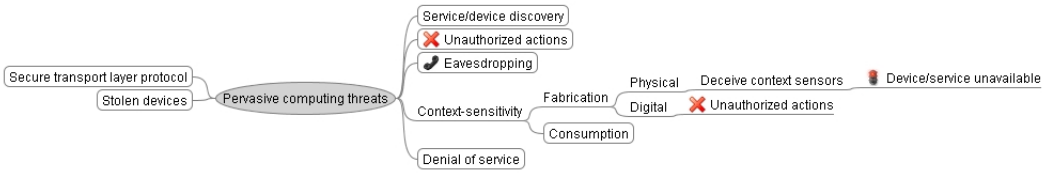

Figure 3.3: Context fabrication threats.

may have a system that tracks when a particular public workstation computer is occupied or free for use and publish this information on a web page. An attacker can provide false physical context to the sensors that the system uses making the workstation always appear to be occupied. Legitimate users that periodically check the relevant web page find the workstation to be constantly engaged. Context fabrication attacks can also target digital context data collected by sensors. By providing such false information an attacker can manipulate a system into granting her authority she was not meant to possess allowing the execution of otherwise protected operations (we examine the rest of the unauthorized actions' sub-tree in a later section of this chapter). As an example consider an attack in which a malicious party generates false context information, possibly by replaying previously captured identity tokens, that a specific person is present in a smart room while in reality he is not. A context-aware system will adapt to facilitate the needs of that person exposing to the attacker services she was not meant to access. In order to address context fabrication threats a pervasive computing security model must have a way to *authenticate* the context captured by its sensors. We refer to this requirement as *context authentication*.

3.1.2 Context Consumption Threats

In context consumption threats an attacker captures context information transmitted in a smart room (summarized in Fig. 3.4). This context information can either be low-level sensor data captured as they are communicated from the environmental sensors to the system, or high-level context data aggregated by the system and captured by the attacker as they are transmitted to legitimate users. This leads to a threat we call *smart space reconnaissance*, which basically is similar to the eavesdropping attack on conventional networks but focuses on learning the context information associated with specific users and

the smart room in general instead of system passwords. The attacker is able to enumerate all devices and services that exist in the environment and use this list to select targets for performing traditional attacks against them, like for example gaining unauthorized access to a device. Smart space reconnaissance can also lead to the invasion of privacy of legitimate users. The attacker can track the movements of a user and always know where he is. Furthermore, personal user information is also disclosed to an attacker that captures context. Details such as the operational parameters of pervasive medical devices are of sensitive nature and their unauthorized possession constitutes a direct violation of the owner's privacy.

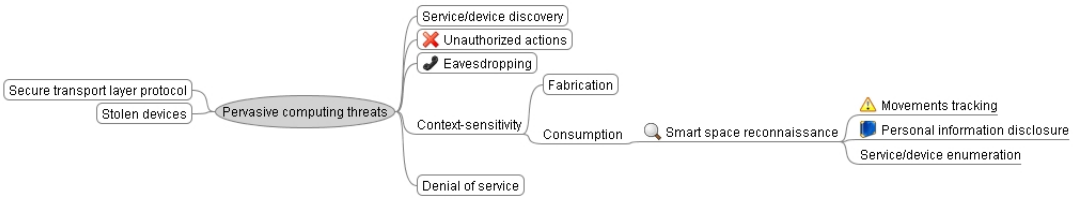

Figure 3.4: Context consumption threats.

3.2 Unauthorized Action Threats

Threats from unauthorized actions (illustrated in Fig. 3.5) may result either from the absence of an authorization system that controls access to the resources that exist in a smart space, or from the successful bypassing of such mechanisms by an attacker. The latter can be accomplished with many different ways, like for example with context fabrication that we have examined in a previous section. Consequently the attacker gains access and is able to use the services of a smart room without the consent or the knowledge of the owner. While this threat has generally the same results as in traditional computing when it comes to digital services, in pervasive computing the attacker can also manipulate environmental controls, like air conditioning settings and the degree of illumination, directly affecting the people that are currently in the room. Moreover, unauthorized invasions of an entity's personal space comprise privacy violations [And01].

In order to address the threats of unauthorized actions access control systems are

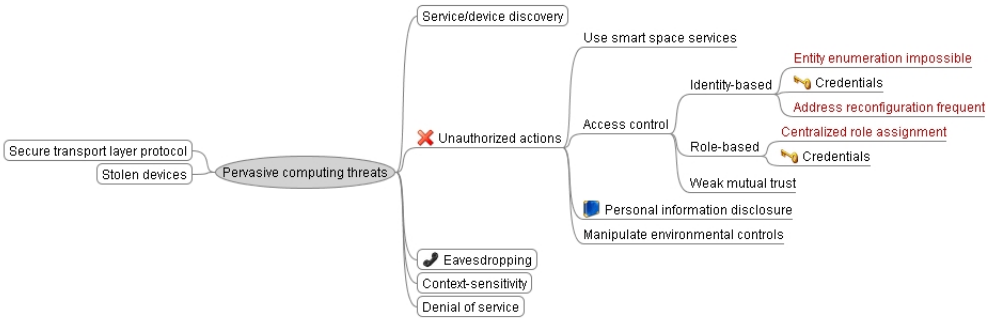

Figure 3.5: Unauthorized action threats.

employed. As we have already discussed, both identity-based and traditional role-based solutions to this problem are incompatible with the requirements of pervasive computing. Moreover, the reliance on credentials to implement such access control mechanisms opens more avenues of security vulnerabilities. Another threat that falls into this category but is not readily apparent concerns the low level of trust that usually exists between the entities of a pervasive computing environment. Users meet each other frequently with little or no previous experience regarding behavior and trustworthiness. Therefore, it is difficult for them to correctly assess a situation and assign the proper access rights to another user or device. This problem becomes even more important when we consider the fact that pervasive computing targets all users and not only the ones that are technologically savvy and can easily understand how to grant specific access rights for specific resources.

3.3 Eavesdropping Threats

Eavesdropping on the communication channel is a common threat for networking systems and protocols that do not employ cryptographic mechanisms to protect the confidentiality of the exchanged messages. An attacker exploits the assumptions of the underlying networking technology, like for example the broadcasting nature of Ethernet, and receives on the local interface the entire traffic of the current subnet. If the confidentiality of the protocol messages is not protected the attacker has access to all the information included in them. In pervasive computing the problem is aggravated by the fact that data transmis-

sions use wireless media, such as Bluetooth [Blu03] and IEEE 802.11b [IEE97]. Therefore, it is far easier for an attacker to simply be within the transmission range of the utilized medium and capture the exchanged data traffic. Moreover, the link layer security mechanism provided as part of IEEE 802.11b, known as Wired Equivalent Privacy (WEP), has been shown to be vulnerable to eavesdropping attacks [BGW01]. A malicious user can recover the shared symmetric key used to ensure the integrity and the confidentiality of the traffic exchanged between the legitimate participants of the network simply by observing data packets.

Another avenue of attack opens when pervasive computing environments are connected to the Internet through gateways. The main protocol to enable this functionality is the Wireless Application Protocol (WAP). WAP allows users to access information on the Internet instantly via handheld wireless devices such as mobile phones, pagers, smart phones and others. This introduces new threats and vulnerabilities for the mobile users and the corresponding service providers. The architecture that is used at the present time is not able to address security problems since it is not based on an end-to-end model. The Wireless Transport Layer Security (WTLS) [WAP01] protocol offers application layer security services to WAP and uses a gateway-based architecture that forces all network traffic to pass through a dedicated system that performs the necessary protocol translations (from WTLS to SSL/TLS, and vice versa) in order to securely connect a mobile device to the Internet. This process breaks the end-to-end security model and allows the gateway to monitor all traffic exchanges. Hence, it becomes a very attractive target for attackers.

There are several threats to a pervasive computing system associated with eavesdropping; the relevant sub-tree of the threat model is shown in Fig. 3.6. One threat we have already identified and analyzed in a previous sub-tree is smart space reconnaissance. The attacker can enumerate the provided services, the exchanged context data, and the context producers and consumers of a smart environment. Moreover, an attacker can monitor and record the link layer identifiers, for example the Media Access Control (MAC) addresses, used by the network interfaces of the users' devices. This leads to user session linking attacks in which the movements of a user from one pervasive administrative domain to the other can be tracked. An assumption of this attack is that link layer identifiers are globally

unique and therefore linkable. This information can be used to stalk an unsuspecting user, register visited locations, and even identify the user's schedule; all of which are of course privacy violations. Finally, by dissecting cleartext protocols an attacker can discover account registration data and other personal information details, like social security numbers for example, that can ultimately lead to identity theft.

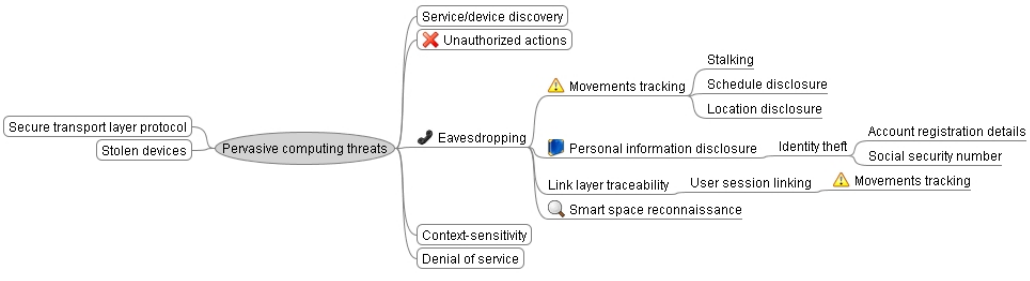

Figure 3.6: Eavesdropping threats.

3.4 Service and Device Discovery Threats

Service and device discovery are an essential part of the pervasive computing paradigm. These technologies enable the on-demand utilization of processing capabilities offered in a smart space without the need for extensive manual administrative tasks. However, they are vulnerable to several attacks (summarized in Fig. 3.7).

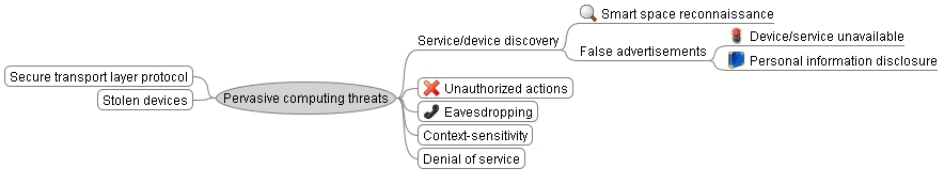

Figure 3.7: Service and device discovery threats.

If there is no security mechanism to protect the confidentiality and the integrity of the service discovery protocol advertisement messages an attacker can capture and record their contents. This leads to the smart space reconnaissance attack we have already examined

in a previous sub-tree. Moreover, if the utilized advertisements are not authenticated according to a security model, an attacker is free to generate and broadcast false messages claiming that he provides a certain service. Legitimate smart space participants looking for such a service would accept and try to access the false one advertised by the attacker. One consequence of this is that the attacker can fully or partially disable the correct operation of any provided service of the environment simply by claiming that he provides the same one and ignoring the incoming access requests. The attacker can also choose to accept the access requests and ask from the user to supply personal information, like passwords or other confidential data, in order to authorize the use of the service. This leads to the disclosure of personal user data to unauthorized parties and is of course a privacy violation.

3.5 Denial of Service Threats

Denial of service threats aim at disrupting the requirement of availability, which can be defined as the problem of enabling systems to perform their advertised services in a timely manner [Gli84]. In pervasive computing denial of service threats can be divided into three categories; battery exhaustion (also known as sleep deprivation torture [SA00]) threats, physical layer jamming, and application layer attacks. These are shown in Fig. 3.8.

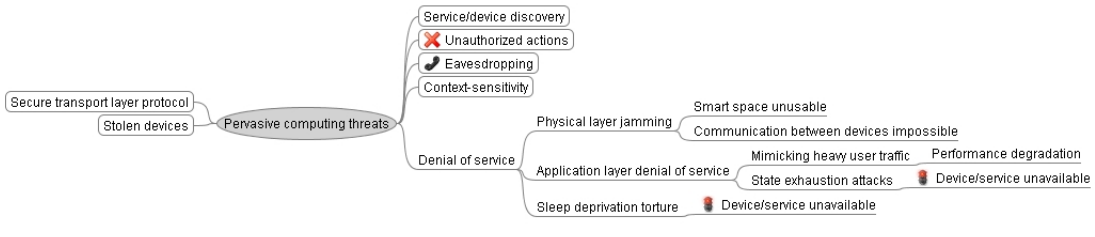

Figure 3.8: Denial of service threats.

The main goal of an attacker that performs a battery exhaustion attack is to render the target service or device unusable. In order to achieve this she constantly tries to involve the victim in expensive dialog exchanges keeping him from entering a sleep state and thus conserving battery energy. Although this type of attack has a very limited impact on traditional computing devices, it is particularly effective on the energy-constrained devices

that are typically used in pervasive computing environments. Physical layer jamming constitutes an attack of a larger scale since it aims at disrupting all the communications taking place in a smart environment. The attacker employs the use of specialized hardware that is able to transmit jamming signals that affect all the channels that the utilized physical layer technology supports. Although the results of such attacks are devastating since they render all communications impossible, they are easily detected. A much more subtle way to perform denial of service attacks is to target the application layer. A motivated attacker can cause the performance of a provided service to degrade to unusable levels by mimicking heavy user traffic. By periodically initiating new access requests from falsely generated sources the attacker can hide from the source provider the fact that an attack is taking place. State exhaustion attacks are also performed at the application layer. They aim to exploit the limited memory resources of embedded and handheld devices by initiating connections and never terminating them. If the target service is not able to handle broken connections efficiently then all its available memory resources are consumed in the effort to maintain the sessions established by the attacker.

3.6 Stolen Device Threats

The devices that are used in pervasive computing are typically small in size, like for example mobile phones and PDAs, and therefore they can easily be stolen. We have identified three threat categories associated with stolen devices; unauthorized actions, installation of unauthorized software, and key material disclosure (see Fig. 3.9).

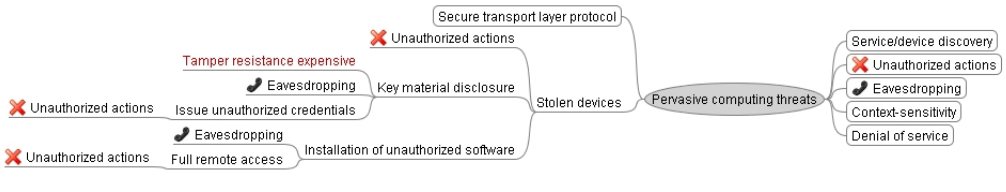

Figure 3.9: Stolen device threats.

A stolen device can be directly used by its physical owner to perform actions in a smart space. This is possible since mobile devices do not generally have security features

to guarantee that they are operated by their rightful owner. Mechanisms like fingerprint identification are slowly being integrated into new generation mobile devices, however they have been proven to be easy to break [MMYH02]. When a device is under the physical control of a malicious entity one of the most common avenues of attack is the installation of unauthorized software that allows the monitoring of all data communications or provides remote access capabilities. Thus, the device falls under the full control of the attacker. Another threat related to stolen devices is the disclosure of key material. Security mechanisms usually rely on symmetric or asymmetric cryptographic keys; these are stored in the participating devices with little or no protective measures. Tamper resistance technologies that can provide some measure of assurance are very expensive and are usually inadequate [AK96], [AK97]. Hence, it is valid to assume that if an attacker physically controls a device he is eventually going to retrieve the cryptographic key material stored in it. With these in his possession he can eavesdrop on protected communication channels and issue unauthorized certificates to principals of his own choosing.

3.7 Secure Transport Layer Threats

One of the most common approaches to secure a networking environment is to develop a protocol that offers authentication, integrity and confidentiality services at the transport layer. This has a number of advantages over network layer security solutions like IPsec [KA98] which are implemented in the operating system kernel making them particularly inconvenient to deploy. Furthermore, IPsec has been criticized for being exceptionally complex and this fact hinders in depth security evaluations [Res00]. Protocols like the Secure Sockets Layer (SSL), the latest version of which is also known as Transport Layer Security (TLS), is by far the most widely deployed security protocol in the Internet [Res00]. TLS utilizes the X.509 identity-based security infrastructure in order to authenticate the peers that want to establish a secure channel. This is achieved through the use of public key digital signatures, the corresponding X.509 public key certificates (PKCs), and the authenticated Diffie-Hellman protocol. The establishment of a secure channel can also be accomplished with other mechanisms like the anonymous Diffie-Hellman protocol and

symmetric shared secrets or username/password/PIN combinations[1] which are vulnerable to different types of threats. In the following subsections we analyze these and other threats to secure transport layer protocol solutions for pervasive computing environments.

3.7.1 Traffic Analysis

Traffic analysis can be defined as the process of intercepting and analyzing exchanged protocol messages in order to derive characteristics of a channel and its participants from patterns in communication [FGBZ03]. It can be performed even when the messages are encrypted and the attacker does not possess the appropriate key material to decrypt them. The characteristics that can be retrieved include the network identifiers of the participants (like IP or MAC addresses), their location based on the identified network segments, the frequency and the duration of the communication, the amount of data exchanged, and others. Depending on the security requirements of a pervasive computing environment such information leaks can be considered to be a substantial threat. Moreover, traffic analysis can result in the tracking of devices as their owners move from one pervasive domain to another. The relevant sub-tree of the threat model is shown in Fig. 3.10.

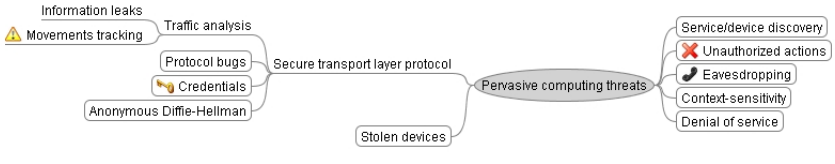

Figure 3.10: Traffic analysis threats.

3.7.2 Protocol Bugs

Another possibility of attacking a secure transport layer protocol lies in possible bugs. These can manifest in different areas of the protocol such as the design logic, the implementation, or the cryptographic algorithms that are used as security building blocks (see Fig. 3.11).

[1] Of course the same mechanisms can also be employed at the network layer. Although we focus on the transport layer as the most popular and convenient way to offer security services, the identified threats

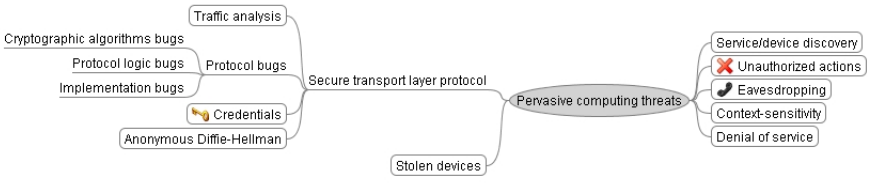

Figure 3.11: Protocol bug threats.

An example of a design flaw is the downgrade attack in SSLv2. The protocol does not provide any protection for the handshake procedure, therefore it is possible for an attacker to force the communicating peers to agree on using a weak cryptographic algorithm even though they both support a stronger one [Res00]. The problem has been solved in SSLv3 by computing a message digest over the entire handshake.

3.7.3 Credential Threats

Credentials are used in order to authenticate and subsequently to authorize users. Based on a successful authentication and authorization result, the utilized secure protocol establishes a protected channel between the participants for further communication. There are different types of credentials that can be used for this purpose; username/password/PIN combinations, shared symmetric cryptographic keys, and certificates. Each credential type suffers from different threats and these are examined in the following paragraphs.

3.7.3.1 Username/Password/PIN Threats

Username/password or PIN combinations can be used in order to authenticate peers to each other and establish secure channels. To use username/password credentials the service provider must create appropriate entries in its authentication database for every user that is trusted and should be granted access. PINs are used by typing the same identification number or string in the two or more devices that wish to communicate. Then this common secret is used as a basis for creating a secure channel. In pervasive computing where the complete set of participating entities is not known in advance and its membership list

apply to network layer security protocols such as IPsec as well.

changes dynamically and frequently these processes present distribution problems. Furthermore, the manual setup of the username/password pairs and PINs is highly obtrusive for the owners of the corresponding participating devices. The main threat with these types of credentials lies in the fact that they are easy to guess[2] and therefore an attractive target for brute force attacks (see Fig. 3.12). Published results in the literature have demonstrated that users rarely select appropriately strong passwords to secure their devices [YBAG04]. PINs are equally vulnerable to attackers; the Bluetooth security mechanism that is based on a PIN system to facilitate the secure pairing of two devices has been recently shown to be vulnerable. The shared PIN can easily be retrieved through brute forcing by a passive attacker that is able to eavesdrop on the pairing process [SW05].

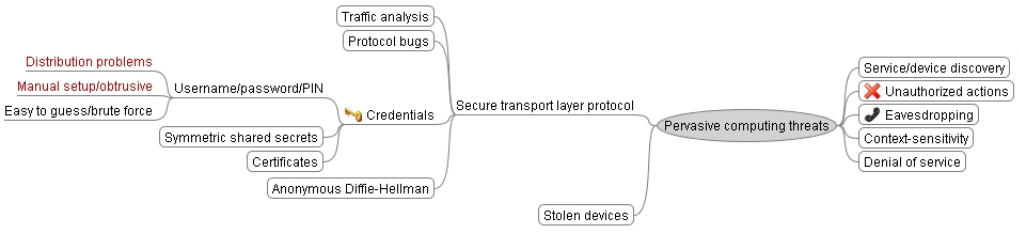

Figure 3.12: Username/password and PIN threats.

3.7.3.2 Symmetric Shared Secret Threats

Another way to establish secure channels is by distributing the same symmetric cryptographic secret key to all of the participants of the environment or of a specific communication channel. This solution is very cumbersome when it comes to managing the shared keys and also requires manual intervention from the human users in order to give the required key material to new participants. The main threat with this solution lies in the fact that each device that has a copy of the shared secret quantity represents a single point of attack for the entire system (see Fig. 3.13). The attacker usually selects the least well protected device and attacks that one to retrieve the secret key. He is then able to attack the system through most of the avenues with have discussed in previous paragraphs.

[2]Since they are chosen to be easy to remember.

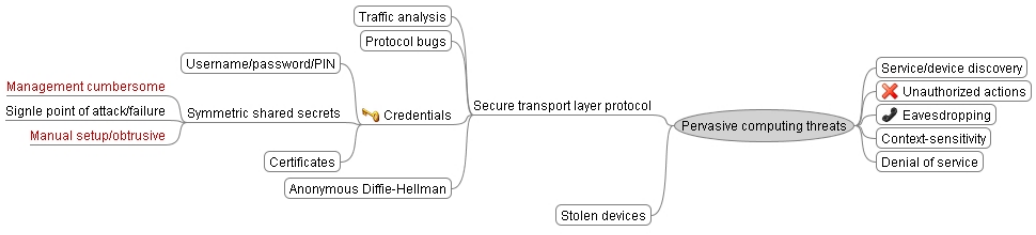

Figure 3.13: Symmetric shared secret threats.

3.7.3.3 Certificate Threats

The threats against the use of certificates to establish secure channels depend on the way they are used towards that end. Certificates can be used as part of a PKI, as self-signed certificates, or as signed statements carrying capabilities. Moreover, we have identified two threats common to all of these categories; public key linking and attacks against revocation mechanisms. The identified certificate threats are summarized in Fig. 3.14.

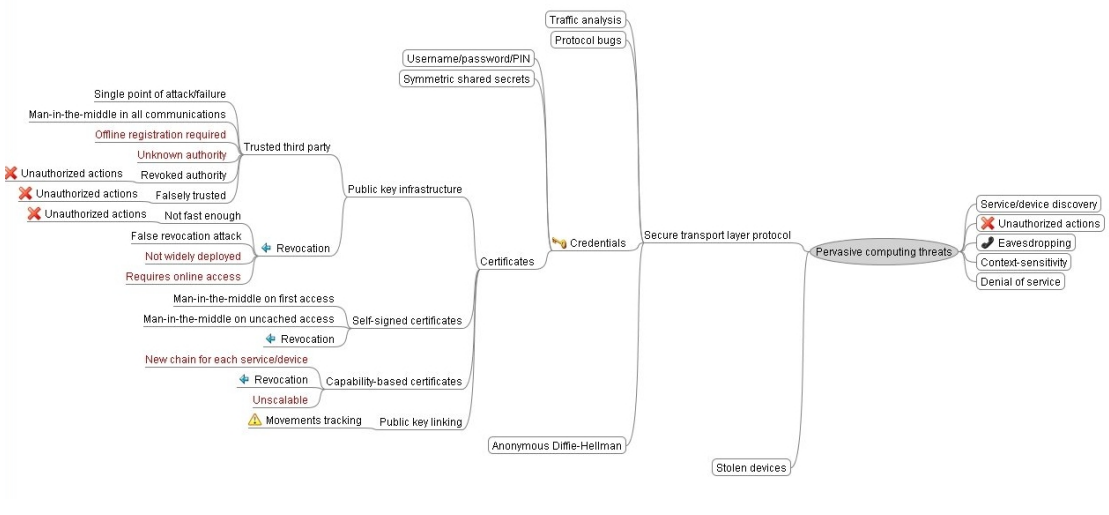

Figure 3.14: Certificate threats.

Certificates are usually signed statements that testify a belief of the issuer, hence they are usually considered to be public in nature. However, this can lead to privacy intrusions. An attacker can monitor the certificates used in different environments, record the public

keys of the subjects and link this information to track the movements of individual users. In PKIs, the authority to issue public key certificates lies entirely with CAs, which are globally trusted third parties and therefore represent a single point of attack for the entire system. This creates a number of additional threats. In the case of a compromised root CA the whole PKI system can be circumvented; unauthorized certificates can be issued, message exchanges can be monitored, and the attacker can perform man-in-the-middle attacks in all communications. Moreover, entities must register and get their public keys certified by the CA beforehand in an offline procedure that requires manual administrative tasks. The bridge trust model of PKI creates another threat when applied to pervasive computing environments. CAs are responsible for different administrative domains and the users that participate in them. When mobile users move from one domain to another the corresponding CAs must trust each other in order for the users to be authenticated in the foreign domain they visit. This implies that the CAs have some method to evaluate each other and cross-certify their keys, and that there exists the required infrastructure to check for revoked certificates at the CA level. The absence of such mechanisms can lead to falsely trusted CAs and the inability to check whether the key-certifying authority of a CA has been revoked or not. The process of revocation itself constitutes another threat avenue related to both PKIs and all certificate-based security protocols. An attacker that has compromised a CA (or any other entity that is able to generate certificates) can freely issue false revocation messages, negating the authority granted to legitimate principals. Another problem with revocation is that it has to be fast enough in order to reach all the users of the revoked certificate before its holder uses it after it has been revoked. This requires constant online access and connectivity between all participants, a requirement that is not compatible with the disconnected nature of pervasive computing. Finally, revocation mechanisms have not been deployed in any great extend and are usually ignored by security engineers.

Self-signed certificates are generated and signed by the principal that is the subject of the certificate; in other words the same principal plays the role of both the issuer and the subject. They are used to allow a principal to testify in a certificate format certain statements about itself. For example, in X.509 self-signed identity certificates a

principal states the binding between its own public key and identifier. Of course since no trusted third party guarantees the authenticity of the information contained in self-signed certificates the issuer is free to create false ones. However, they are usually used in application environments where always online central entities such as CAs cannot be assumed, or are too expensive to implement. If a service provider tries to establish the identity of a requesting entity based on a self-signed certificate then a man-in-the-middle attack is possible. The attacker takes advantage of the fact that the service provider has no way of verifying the binding between the identifier and the key of the requester. For example, the Secure Shell (SSH) protocol [Ylo96] tries to avoid relying on a PKI by using self-signed certificates and establishing the binding between the identifier and the key of service providers by assuming that the first time the connection happens no attacker substitutes the provider's key with his own. This practice makes the protocol vulnerable to a man-in-the-middle attack on the first access. Furthermore, if the service requester does not cache the bindings between identifiers and keys the same attack is also possible on subsequent access attempts.

Capability-based certificates, like the ones used in the KeyNote and SPKI/SDSI systems, also suffer from the threats related to revocation we have already analyzed. As it has been previously discussed, they are not compatible with the requirements of pervasive computing since they involve the creation of a new delegation chain for each new service provided in an environment and the distribution of the related credentials to actually implement the chain. This leads to scalability problems and assumes that the human owners of the devices will perform the necessary configuration tasks to issue and distribute the certificates of a chain.

3.7.4 Anonymous Diffie-Hellman Threats

Any security mechanism that relies on the original Diffie-Hellman protocol for establishing protected communication channels suffers from man-in-the-middle attacks (see Fig. 3.15). Since there is no way for the participating principals to prove to each other their identities an attacker can compromise the secrecy of the established key. Another possible attack is known as the *identity misbinding attack* [BM03]. The attacker does not aim to compromise

the secrecy of the established key, but the authenticity of the exchange. Any traffic coming from a legitimate requester appears to the service provider to have originated from the attacker.

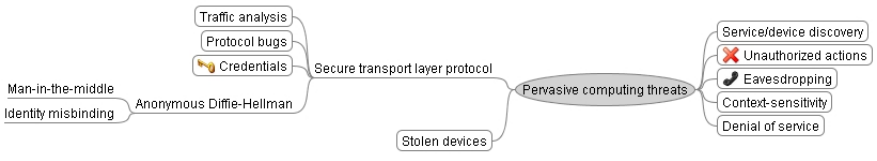

Figure 3.15: Anonymous Diffie-Hellman threats.

Chapter 4

ÆTHER Architecture

ÆTHER is an authorization management framework for pervasive computing. It allows the owners of pervasive devices to locally define access control policies without the need of contacting outside centrally managed authorities. Furthermore, ÆTHER supports disconnected operation, context-awareness and follows an unobtrusive usage model. The configuration tasks that are needed in order to establish authorizations between the users are gracefully integrated with the performance of the primary physical tasks minimizing distractions.

In this chapter we first present the part of the previously analyzed threat model that is addressed by ÆTHER. Initially we focus on the presentation of the general ÆTHER framework as an extension to the core RBAC model. We instantiate our authorization management framework into two different architectures, namely ÆTHER_0 and ÆTHER_1, that address the security and the user demands of different Pervasive Authority Domains (PADs). A PAD is an abstraction that organizes policy and privilege within a domain and is defined differently in our two instantiations. Although both instantiations of our model satisfy the previously identified requirements, they do so by employing different cryptographic mechanisms, design approaches and management models.

The ÆTHER framework, according to the categories defined by Chen and Kotz in [CK00], follows the active context-awareness category. Therefore, it is able to dynamically adapt its behavior based on sensed contextual information from the environment. This ability relates to the reached authorization decisions, and the context utilized is both that

of the requesting entity and that of the service provider. For example, ÆTHER allows the specification of context-aware policies that allow the execution of a specific action to entities that are in the same physical location as the service provider. If the service provider moves to a new location the policy does not need to be modified; it adapts and operates in the new location as well.

After presenting the ÆTHER framework, ÆTHER_0 and ÆTHER_1, we analyze the usability and maintainability aspects of our architectures focusing on making them easy to be adopted by end users. We also discuss the privacy considerations associated with both our general model and its two instantiations. The chapter closes with a comparison of the two instantiations and an examination of which one is applicable to which pervasive computing scenario.

4.1 ÆTHER Threat Model

Our security model aims to address the threats associated with authorization management both from a technical and a usability perspective. Consequently, we consider several classes of threats identified in the previous chapter to be outside the scope of our research area. For example, denial of service threats assume that the attacker is a properly authorized user of the system, or of the communication medium when the attack focuses on the network or physical layers, while we concentrate on authorization itself. Moreover, we do not consider the threats, although we discuss the repercussions, of stolen devices since we believe that when an attacker has physical access to a device she can eventually bypass all security mechanisms and recover confidential data and key material. Another assumption we make is that the sensors that are embedded into the environment collect and report trustworthy contextual information, that is we are not concerned with context authentication. However, we protect against unauthorized context consumption. Fig 4.1 shows only the threats (nodes) of the general pervasive computing threat model that the ÆTHER security architecture aims to protect against.

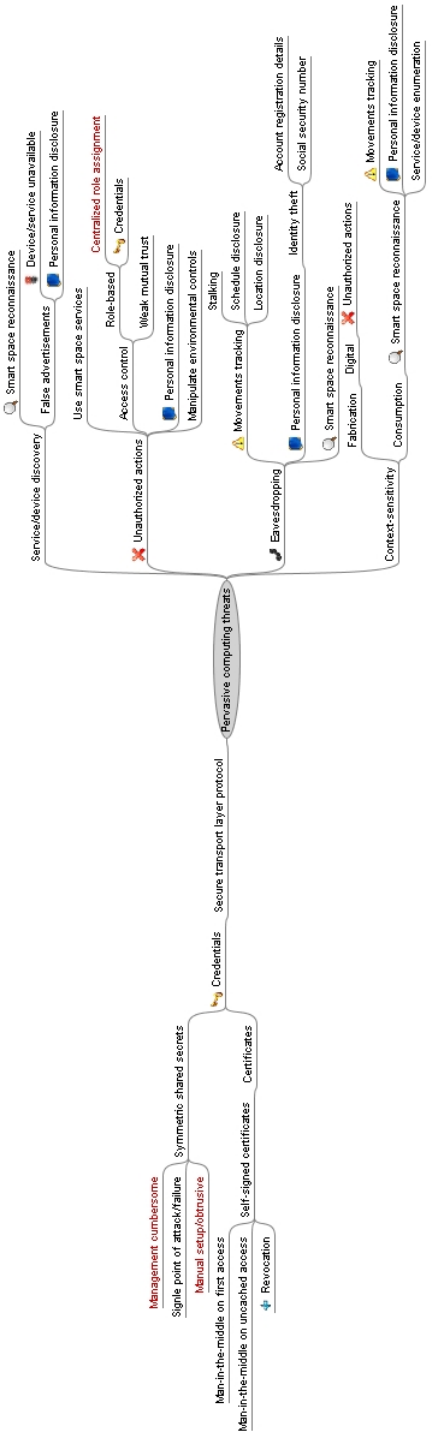

Figure 4.1: ÆTHER threat model.

4.2 General ÆTHER Model

This section examines the general ideas behind our authorization model for pervasive computing. We first present our high-level user-based management model, and we analyze the way it expands the traditional view of establishing and managing security associations. We then focus on defining the ÆTHER model, its extensions to RBAC and its context-awareness features, presenting the associated policy statements and the utilized authority revocation mechanism.

4.2.1 User-based Authorization Management

In pervasive computing systems users and their requirements are placed at the center of every design choice. The ultimate goal is to develop systems that are first of all usable, meaning that they fulfil the expectations of their users, and unobtrusive in the sense of not distracting users from their primary physical tasks. We believe that a pervasive computing system's security needs are included in the usability requirement. For example, when a user decides to share a document stored on his PDA with a trusted colleague, he expects that the system will ensure the confidentiality of the exchange so that a nearby malicious entity cannot access it. Traditional security protocols rely on digital data, like credentials and policies, to establish secure associations and reach access control decisions. However, in pervasive computing, human users are directly involved in the performed tasks as these occur in the physical space. Each participating user has a mental model formulated in her mind that represents the particular circumstance of the task at hand. This perceptive model includes the following components: The other users and the role they play in the occurring event, the resources provided for controlled access, and the relationships between all of these components. The system must allow the user to express this perceptive model of the situation to the pervasive computing devices in order to define security relationships and this must be accomplished in an unobtrusive manner. We believe that this expression of the users' mental security model can be integrated gracefully with the performance of the primary task at hand. Continuing the example with the shared file, the owner can state his intent that he trusts the person next to him by pointing his PDA to his colleague's handheld device. The authorization data, like credentials, necessary for securing the exchange can

be established through this physical action that states the intent of the user. Fig. 4.2 illustrates the differences between the traditional authorization management approach and our own *user-based* authorization management model for pervasive computing. While the traditional model is concerned only with the protocols between the devices and the static security policies, the ÆTHER management model also includes dynamic policies inferred from the physical actions of the users, each playing a specific role.

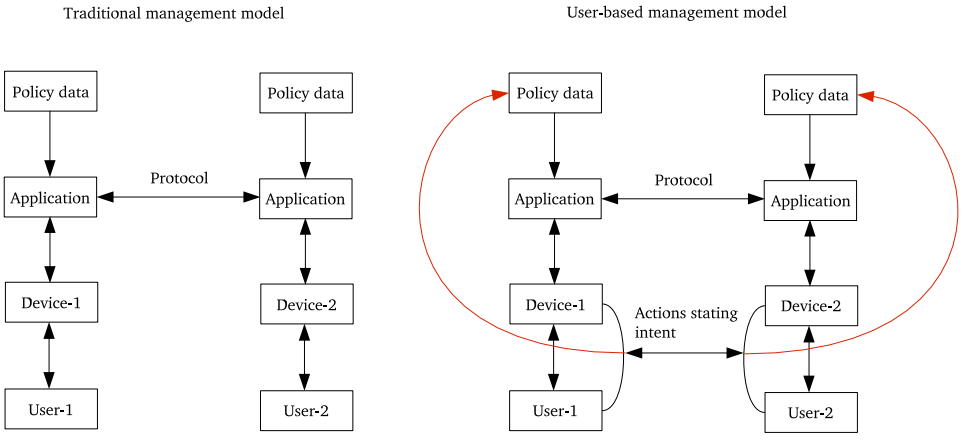

Figure 4.2: Traditional vs. user-based authorization management.

We believe that the concept of location-limited channels as first defined by Stajano and Anderson [SA00] and later extended by Balfanz et al. [BS02] can be used to establish policy data, like for example the issuing of credentials from one user to another, and be integrated gracefully to the users' physical actions. These actions are part of the effort a user is already undertaking in order to accomplish the primary task and they state the intent of the user regarding authorization. Therefore, we accomplish unobtrusive user-based authorization management that does not distract users with complicated security-related dialogs, decisions or creation of new accounts.

4.2.2 ÆTHER Model

The general ÆTHER model extends the core Ferraiolo-Kuhn RBAC model to provide authorization management for pervasive computing environments. While RBAC depends on a fully centralized management model in which a system administrator is responsible

for user and permission assignment, ÆTHER utilizes the user-based model described in the previous section. This high-level model is instantiated into two distinct architectures, namely ÆTHER$_0$ and ÆTHER$_1$. We believe that the decentralized and highly dynamic nature of pervasive computing is compatible with authorization relationships based on roles, given that role authority does not follow a centralized management model. This requirement for decentralization relates both to the ability to make assertions concerning user assignments to roles and the delegation of authority over a specific role.

The ÆTHER model makes the following semantic distinction between subjects and users. A user may have multiple subjects in operation. Each subject is uniquely referenced by an identifier (a message digest of the hosting device's hardware addresses concatenated with a randomly generated number in ÆTHER$_0$ or a public key in ÆTHER$_1$). There is no way for the system to map subjects to the identities of physical users, as is the case in RBAC, thus providing a layer of privacy. Moreover, ÆTHER uses *attributes* instead of roles. An attribute is defined as a property of an entity, while a role is a type of attribute used to define the position an entity has in an organization. The ÆTHER model is composed of the following components (a graphical representation of the components and their relationships is shown in Fig. 4.3):

- A set of subjects: S
- A set of resources: R
- A set of operations: O
- A set of authority attributes: A_A
- A set of context attributes: C_A (‡)[1]
- A set of resource attributes: R_A (‡)
- A set of permissions: $P = \mathcal{P}(O \times R)$
- Authority attribute to subject assignment: $SA_A \subseteq A_A \times S$
- Context attribute to subject assignment: $SC_A \subseteq C_A \times S$ (‡)

[1]The (‡) mark denotes an ÆTHER extension to the core RBAC model.

- Resource attribute to resource assignment: $RA \subseteq R_A \times R$ (‡)
- Basic permission to authority attribute assignment: $PA_A \subseteq P \times A_A$
- Basic permission to context attribute assignment: $PC_A \subseteq P \times C_A$ (‡)
- Basic permission to resource attribute assignment: $PR_A \subseteq P \times R_A$ (‡)
- Composite permission to authority attribute assignment: $XPA_A \subseteq PR_A \times A_A$ (‡)
- Composite permission to context attribute assignment: $XPC_A \subseteq PR_A \times C_A$ (‡)
- A function mapping an authority attribute to a set of permissions: $permission_A = A_A \rightarrow \mathcal{P}(P)$, or more formally: $permission_A(a : A_A) = \{p : P \mid (p, a) \in PA_A\}$
- A function mapping a context attribute to a set of permissions: $permission_C = C_A \rightarrow \mathcal{P}(P)$, or more formally: $permission_C(c : C_A) = \{p : P \mid (p, c) \in PC_A\}$ (‡)
- $permission_{XA} = A_A \rightarrow \mathcal{P}(PR_A)$, or more formally: $permission_{XA}(a : A_A) = \{p : PR_A \mid (p, a) \in XPA_A\}$ (‡)
- $permission_{XC} = C_A \rightarrow \mathcal{P}(PR_A)$, or more formally: $permission_{XC}(c : C_A) = \{p : PR_A \mid (p, c) \in XPC_A\}$ (‡)
- The mapping of a subject to a set of authority attributes: $subject\_authority(s : S) \rightarrow \mathcal{P}(A_A)$, or more formally: $subject\_authority(s : S) \subseteq \{a \in A_A \mid (a, s) \in SA_A\}$
- The mapping of a subject to a set of context attributes: $subject\_context(s : S) \rightarrow \mathcal{P}(C_A)$, or more formally: $subject\_context(s : S) \subseteq \{c \in C_A \mid (c, s) \in SC_A\}$ (‡)
- The mapping of a resource to a set of resource attributes: $resource\_attributes(r : R) \rightarrow \mathcal{P}(R_A)$, or more formally: $resource\_attributes(r : R) \subseteq \{ra \in R_A \mid (ra, r) \in RA\}$ (‡)
- Access authorization: A subject s can perform an operation o on a resource r only if:

1. There exists an authority attribute a that has been assigned to subject s and there exists a permission p that is assigned to a such that the permission authorizes the performance of o on r, or

2. There exists a context attribute c that subject s possesses and there exists a permission p that is assigned to c such that the permission authorizes the performance of o on r, or

3. There exists an authority attribute a that has been assigned to subject s and there exists a resource attribute ra that has been assigned to resource r and there exists a composite permission ap that is assigned to a such that the permission authorizes the performance of o on ra, or

4. There exists a context attribute c that has been assigned to subject s and there exists a resource attribute ra that has been assigned to resource r and there exists a composite permission ap that is assigned to c such that the permission authorizes the performance of o on all resources that have the resource attribute ra.

5. Formally: $access : S \times O \times R \rightarrow BOOLEAN$, $s : S$, $o : O$, $r : R$, $access(s, o, r) \Rightarrow$
   $\exists\, a : A_A$, $c : C_A$, $p : P$, $ra : R_A$, $ap : PR_A$,
   $((a \in subject\_authority(s) \wedge p \in permission_A(a) \wedge (o, r) \in p) \vee$
   $(c \in subject\_context(s) \wedge p \in permission_C(c) \wedge (o, r) \in p) \vee$
   $(a \in subject\_authority(s) \wedge ap \in permission_{XA}(a) \wedge ra \in resource\_attributes(r) \wedge$
   $(p, ra) \in ap) \vee$
   $(c \in subject\_context(s) \wedge ap \in permission_{XC}(c) \wedge ra \in resource\_attributes(r) \wedge$
   $(p, ra) \in ap))$

We have four different kinds of attributes; *authority* attributes, *context* attributes, *aggregated context* attributes, and *resource* attributes. An attribute is defined as an ordered 2-tuple of the form *(name, value)* where *name* is the identifier of the attribute and *value* is its value. Authority attributes are the traditional security attributes used to describe the properties of subjects. These properties are relatively static, meaning that their values are obtained during the establishment of a session. Authority attributes are similar to the

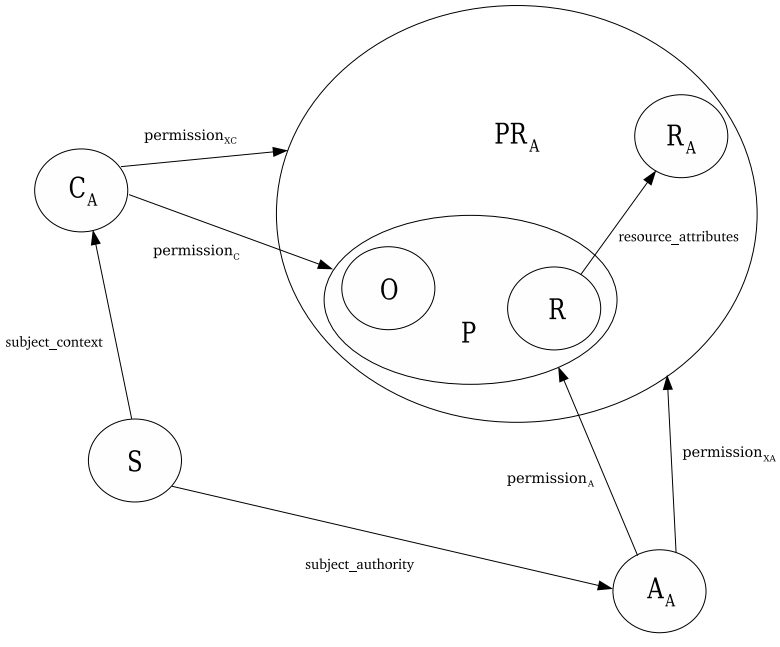

Figure 4.3: The ÆTHER model.

concept of roles in traditional RBAC. A subject is a member of a role if and only if it has been assigned the authority attribute that represents the corresponding role. The values of authority attributes must be configured through manual human intervention and are then obtained from a subject's credentials. In that sense, authority attributes are static security metadata whose values can be used in order to reach authorization decisions. On the other hand, context attributes describe the dynamic properties of subjects associated with their current context. Although manual intervention is still required to designate the use of context attributes in authorization decisions, their values are obtained dynamically every time an access request is made. In contrast to authority attributes, the values of context attributes may change during the lifetime of an active session since they reflect a subject's environmental context. An aggregated context attribute is a context attribute defined as an aggregation of one or more context attributes. The aggregation is performed by a Context Aggregator (CA) that is defined as the following function:

$$CA : C_{A_1} \times C_{A_2} \times ... \times C_{A_v} \rightarrow C_A,\ v \geq 1$$

Resource attributes are assigned to resources to simplify their administration and the

definition of authorization policies. Basic permissions are assigned to resource attributes and define composite permissions. Authority or context attributes are then associated with composite permissions in order to allow the creation of authorizations addressing a whole set of resources. For example, the context attribute *(location, office_008)* can be associated with all output devices allowing anyone in office 008 to access all devices that have been defined as output devices.

The concept of context attributes provides a high-level interpretation of the low-level context information collected by sensors embedded into the environment. The assignment of context attributes to subjects is maintained dynamically based on the raw data of context dissemination sensor service infrastructures, like for example the ones presented in [PLF+01] and [SDA99]. Thus, whenever an access control decision is required that has been defined using a context attribute, the corresponding service is queried regarding the status of the requesting subject, *pulling* for context data.

In ÆTHER permissions are modeled as the rights of authority or context attributes. We associate rights with actions, so possession of an authority or context attribute permits the corresponding principal to perform a certain action. Authorization policies define context attribute requirements or restrictions in addition to the ones using authorization attributes (see Fig. 4.4). This allows the specification of context-adaptive policies that control access to protected resources. For example, the owner of a printer may allow all people physically present in the same room as the printer to use it. However, the information regarding associations between subjects and context attributes may be considered sensitive or private. In these cases we view the context-dissemination service as a normal resource to be protected and accessed only by properly authorized principals. The authorization can be handled by authorization attributes or even context attributes which are not considered sensitive in certain situations. Furthermore, each device maintains its own current context in what we call a Local Context Profile (LCP). When the value of a context attribute is defined as a *dynamic local context value* then this value is retrieved from the corresponding context attribute stored in the device's LCP.

ÆTHER uses this *pull-based* context dissemination approach since we believe that it is much more appropriate for security applications than the *push* approach. In the *push*

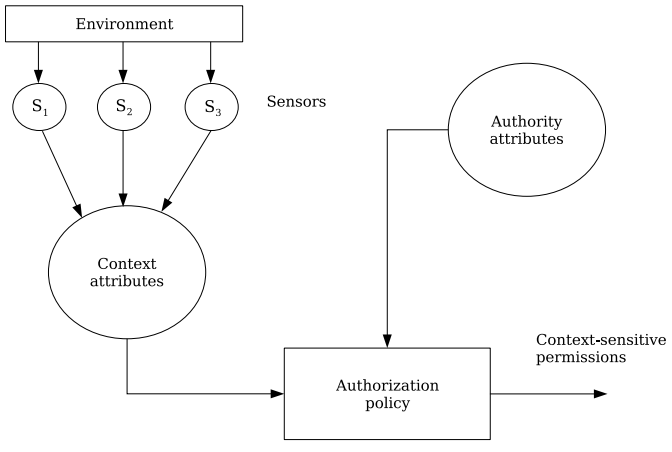

Figure 4.4: Context-sensitive permissions.

approach principals that are interested in specific contexts subscribe to context producers (i.e. sensor devices) and receive updates when the state of the monitored data changes. The main problem with this design choice is that it requires the producer and the consumer to always be within the transmission range of each other since context data can be pushed at any time. Otherwise, the consumer is not able to receive updates and may have an outdated view of the context data. When these data are used to reach security-related decisions, as is the case in ÆTHER, outdated context snapshots may lead to authorizing requests that should have been denied.

The authorization process in ÆTHER works in a similar way as existing traditional trust management systems. A principal makes an access request to a service provider. The validating principal passes the requester's credentials[2] along with the local policy statements and the request to an inference engine that outputs whether the access request is allowed or denied.

We believe that capability-based security systems, like KeyNote, are not able to address the decentralized nature and the requirement for unobtrusive management of pervasive computing. In order to compare such approaches to ÆTHER consider the following example. A home owner wants to allow visitors to her house to control the television set. Capability-based systems lack the ability to define such a policy expressing that any princi-

[2]The requesting principal's credentials are collected differently in the two instantiations of our general model.

pal that is a visitor is allowed to access the television's functionalities. Instead they can be used to delegate the television access right from the public key of the owner to the public key of a visitor's device. However, a ubiquitous computing household may provide tens, or even hundreds, of services. For each provided service the owner wishes to allow a visitor to access she has to issue a delegation. Therefore, capability-based approaches introduce extra administrative overhead for the owner of devices that offer services. In ÆTHER the owner has to specify a *single time* that a principal is a visitor, and then this principal can access all services associated with the *(group, visitor)* authority attribute without further administrative actions from the owner.

4.2.3 Dual Interface and Policy Distribution

The main assumption of the ÆTHER model is that each participating pervasive information processing device satisfies the dual interface requirement. One of the device's communication interfaces must implement a location-limited channel, like contact or infrared, that guarantees authenticity and confidentiality. This channel is used to bootstrap or prime security relationships between devices and requires human interaction for its establishment. The human owners of devices are required to perform certain actions that are gracefully integrated to the general task they are trying to achieve, create location-limited channels and exchange authorization policy information over them. This exchange is by definition secure since location-limited channels ensure the authenticity and confidentiality requirements. Using this bootstrapping data their devices are then able to establish secure communication channels over their primary networking interfaces, which usually implement a wireless technology like for example Bluetooth or IEEE 802.11b, and perform interaction protocols. As we have already discussed in a previous paragraph, the required human actions for the establishment of a location-limited channel and the protocol over a wireless interface between pervasive devices follow our user-based management model.

An obvious question at this point is why the location-limited interface of a device is not used for both the exchange of bootstrapping data and the completion of interaction protocols. Location-limited channels require human intervention in order to be established. While we can integrate the necessary actions for their establishment in the primary task

a user is trying to achieve to exchange bootstrapping data, this can only be accomplished because these actions are simple. If we were to use location-limited channels for the entire protocols then the required actions would be extensive and their integration to the primary task impossible. Consequently, such a solution would be obtrusive to the end users.

The distribution of policy data in ÆTHER is achieved by utilizing both interfaces. When a device is first purchased it is completely free of policy information. The first action that needs to be performed by the owner is to *bind* this new device to one of his old devices. The bind operation is similar to the imprinting of the Resurrecting Duckling system. By establishing a location-limited channel between the two devices, the owner specifies that the new device is to be bound by the old device. The new device now recognizes the old one through the binding policy information they have exchanged. The owner can also choose to bind the new device to more than one master devices, again by establishing a location-limited channel between the two devices and stating that he wants to create a binding. Further policy data, like for example authorizations and authority or context attribute definitions, can be embedded into the new device via the normal primary wireless interface. Not all such policy data are accepted, but only those that are issued by one of its masters. The device can authenticate the issuer of the new policies via the information established during the binding procedure. The assignment of authority attributes in ÆTHER is accomplished over the location-limited channel interface. When a user wishes to issue an authority attribute credential to another user they use their devices to establish such a channel and the issuer states her intent by performing the necessary action. As we have seen, the assignments of context attributes do not require human interaction, but are maintained dynamically by sensor devices that collect and disseminate context. The attributes that an entity possesses are used to support access requests to entities that provide services in the environment.

Finally, we assume that every device provides some physical way (like a switch for example) to delete all the policy state information it holds and return to its original unbound state. This is useful in the case an owner wants to sell, or give away to someone else, a device. Obviously this assumption makes ÆTHER pervasive computing devices attractive to steal. However, even without this convenient functionality a typical device can easily

be dissected and its state information retrieved or deleted [AK96], [AK97].

4.2.4 Policy Statements

In this section we describe the policy statements of the general ÆTHER model. The policy language we use is similar to the assertion language of the KeyNote trust management system. However, we have extended the syntax in order to allow the use of ÆTHER-specific policy data structures. We have to note at this point that this section presents the general policy statements of our architecture. The two instantiations of the general model that will be presented in the following paragraphs analyze these statements and their applications in greater detail. Another important note regards the utilized namespace. We believe that given the distributed and decentralized nature of pervasive computing it would be difficult to enforce a global namespace. Instead ÆTHER follows the design approach that is advocated by SPKI/SDSI and relies on local namespaces. The vocabulary for defining policies (either for the data, the operations, or the conditions) is outside the scope of the work presented in this thesis. Existing research efforts to create ontologies for authorization management and pervasive computing can be used for this purpose [Don03], [CFJ03].

According to the policy model defined by the Internet Engineering Task Force (IETF), a policy is a rule that specifies the execution of certain actions when a set of conditions evaluates to true [MESW01]. This *event-condition-action* model has been primarily defined for managing the Quality of Service (QoS) experienced by network entities like applications and users. Therefore, the events that trigger policy rules are usually related to types and volumes of traffic flows. The ÆTHER policy model directly relates to the IETF event-condition-action model assuming that an implicit event such as a user request triggers a policy rule.

The binding policy statement serves the purpose of binding the subject principal to the issuing principal. This is similar to the imprinting functionality of the Resurrecting Duckling system,

aether version: <The integer 0 or 1 denoting the instantiation of the general model>
type: binding
issuer: <The identifier of the issuer encoded in hexadecimal and put in double quotes>

subject: <The identifier of the subject encoded in hexadecimal and put in double quotes>

The binding between the issuer and the subject principals is enforced differently depending on the instantiation of the general model. The same is true for the utilized identifiers of the principals. Attribute assignments implement the assignment of authority attributes to principals:

aether version: <Integer>
type: attribute assignment
issuer: <Quoted hexadecimal string>
subject: <Quoted hexadecimal string>
attribute name: <String>
attribute value: <String, integer, or float>
not valid before: <Year/month/day-hour:second>
not valid after: <Year/month/day-hour:second>
renewable: <The integer 1 or 0 respectively denoting whether the credential is renewable or not>

Resource attribute assignment statements are used to assign resource attributes to principals:

aether version: <Integer>
type: resource attribute assignment
issuer: <Quoted hexadecimal string>
subject: <Quoted hexadecimal string>
resource attribute name: <String>
resource attribute value: <String, integer, or float>

ÆTHER supports both positive and negative authorizations. Positive authorizations define the required attributes (authority and/or context) that a principal must possess in order to be able to access the specified operation on the specified resource. Negative authorizations are required to support resolving conflict of interests and separation of duty problems which are typical in access control systems. For example, negative authorizations allow the expression of policies such as "a set of under aged principals (e.g.: principals that are mapped to all children under the age of 12) do not have access rights to switch on the television set",

aether version: <Integer>
type: positive authorization (or negative authorization)
issuer: <Quoted hexadecimal string>

resource: <String>
operation: <String>
requires: <Predicates that operate on authority and/or context attributes>

The following notation is used in the specification of the *requires* field:

- @: An authority attribute, example: *(@group, family_member)*.
- $: A context attribute, example: *($location, office_008)*.
- %: An aggregated context attribute, example: *(%activity, meeting)*.
- *: A resource attribute, example: *(*device, input)*.
- _: A dynamic local context value, example: *($location, _location)*.

An ÆTHER$_0$-specific negative authorization policy implementing the above example with the television set is the following:

aether version: 0
type: negative authorization
issuer: "2525554c54f1e32b85bbf827240"[3]
resource: tv_set
operation: switch_on
requires: (@age < 12);

The support of negative authorizations can lead to conflicts when two or more authorizations of different types have been defined for the same resource. The problem of resolving such conflicts has been explored extensively in the computer security literature [LS99]. The approach that we take is to allow the end user to define which authorization type supercedes the other. The user defines a parameter that is global for a particular device's inference engine and gives precedence to either positive or negative authorizations over the other one whenever conflicts arise.

The fields *resource* and *operation* are used to abstract the application-dependent syntax and semantics of the different resources that a principal may wish to protect. ÆTHER does not interpret the semantics of these two strings, it only uses them to identify the

[3]The hexadecimal strings are not shown in their entirety for readability reasons.

authorization policies that are associated with the application resource. Pervasive computing developers that use ÆTHER to secure access to their applications have to agree on the semantics of the strings utilized for referring to resources and operations. Hence, the exact resource descriptions for a device's access interface are outside the scope of ÆTHER. Developed standards such as the Universal Remote Console (URC) [LTZV04] can be used for this purpose.

Another type of policy statement is the one that implements the assignment of permissions to resource attributes:

aether version: <Integer>
type: permission resource attribute assignment
issuer: <Quoted hexadecimal string>
resource: <String>
operation: <String>
resource attribute name: <String>
resource attribute value: <String, integer, or float>

Based on these assignments we can specify composite positive and negative authorizations that define the authority or context attributes a principal must have to gain the access rights associated with specific resource attributes. Composite authorizations allow us to express policies that for example grant access to all output devices in the environment to any principal that has been certified as a visitor or is in the same location as the target output device,

aether version: <Integer>
type: composite positive authorization (or composite negative authorization)
issuer: <Quoted hexadecimal string>
resource attribute name: <String>
resource attribute value: <String, integer, or float>
requires: <Predicates that operate on authority and/or context attributes>

As an example consider the case of an audio player (an output device). An ÆTHER_1 composite positive authorization statement for all output devices could be the following:

aether version: 1
type: composite positive authorization
issuer: "3048024100cd462d1d7cf0aa5043"
resource attribute name: device

```
resource attribute value: output
requires: (@group == visitor) || ($location == _location);
signature: "7c010f91d2240c6884e30203"
```

After the audio player's owner assigns to his device the resource attribute *(device, output)* with a resource attribute assignment statement, he also has to define what are the permissions associated with the audio player as an output device,

```
aether version: 1
type: permission resource attribute assignment
issuer: "3048024100cd462d1d7cf0aa5043"
resource: audio_player
operation: play_track
resource attribute name: device
resource attribute value: output
signature: "d37fd9efeefb2063cdce8b"
```

Now any principal certified with the authority attribute *(group, visitor)* or that is in the same physical location as the audio player can play a track on it.

The principals that are responsible of maintaining the associations between context attributes and principals are specified in policy statements called Context Attribute Sets (CASs). Such principals are sensor devices and through CAS statements are given authority over a specific context attribute:

```
aether version: <Integer>
type: context attribute set
issuer: <Quoted hexadecimal string>
attribute name: <String>
attribute value: <String, integer, or float>
sources of authority: <List of comma separated and quoted hexadecimal strings>
```

Sensor devices maintain the assignments of context attributes to principals using *dynamic sets* defined for specific context attributes. A normal non-dynamic set can be described according to set theory as a static assortment of elements. On the other hand, a dynamic set can be described as an assortment of elements that varies with respect to time. Hence, a non-dynamic set can be viewed as an instance of a dynamic set in a specific time point. We formally define a dynamic set as follows:

Let V be a set of principals and $\mathcal{P}(V)$ the power set of V. Let T be a time period, i.e. a set consisting of specific points in time. We define the function $f : T \rightarrow \mathcal{P}(V)$ as a dynamic set that consists of elements that belong to V in the time period T.

For example, if we want to allow access to a provided resource only to entities that are in the same location as the provider we could create a context attribute (defined as f) for this location, e.g. the context attribute *(location, office_008)*. The elements of this dynamic set at time $t_1 \in T$ would be given by $f(t_1)$. In other words, a principal $v_1 \in V$ would be allowed access if and only if the statement $\exists t_1 \in T \bullet v_1 \in f(t_1)$ holds.

The final policy statement of the general model is the one that defines context aggregators:

```
aether version: <Integer>
type: context aggregator
issuer: <Quoted hexadecimal string>
input: <Predicates that operate on context attributes>
output context attribute name: <String>
output context attribute value: <String, integer, or float>
```

As an example consider the following ÆTHER$_0$ context aggregator for the aggregated context attribute *(state, sleeping)*:

```
aether version: 0
type: context aggregator
issuer: "778f13d39be32b86be92749ca"
input: ($location == bedroom) && ($heart_rate < 60) && ($inactivity_period > 60);
output context attribute name: state
output context attribute value: sleeping
```

4.2.5 Authority Revocation

In certain occasions the authority given with an issued credential needs to be revoked. Such occasions may include loss or compromise of the key used to issue a credential, loss or compromise of the key to which a credential has been issued to, or realization on the part of the issuer that the information contained in a previously issued credential is no longer valid. There are four mechanisms to achieve the revocation of a credential:

1. By using the validity period of the credential itself and setting it to a short time period. This mechanism effectively limits the window of revocation to the validity period specified as part of the credential. However, its use presents several problems. If the validity time period is too short then it can result in frequent re-issuing of credentials involving the issuer in constant obtrusive administrative tasks, or time periods during which access to a particular service is impossible. On the other hand, if the validity period is set to a high value then a required revocation may not take place until this period expires.

2. Another mechanism is the use of Certificate Revocation Lists (CRLs), which are basically lists that include all credentials that have been revoked. The issuer must create a CRL and distribute it to all participants of the system. There are two main problems with CRLs. The first one has to do with the distribution of the list to all interested parties, which can take place either with flooding techniques or with placing the list on a public repository from which it can be retrieved. Both approaches require periodic access to the issuer; as the period that the issuer cannot be contacted increases so does the associated risk of the verifier. The second problem is scalability; as the number of revoked credentials increases CRLs become more cumbersome to manage.

3. Current Identity Lists (CILs) follow the opposite approach of the CRLs mechanism. CILs are signed lists that include all the valid credentials that have been issued by a given credential authority. Obviously, in environments with large numbers of participating entities CILs can grow to become cumbersome to manage efficiently.

4. The final revocation mechanism is the one advocated by OCSP [MAM+99] and requires the credential verifier to check the revocation status of a credential at every verification. This basically means that the verifier must always be able to contact the issuer of every credential that is presented to him. Furthermore, since the verifier is also a service provider the process of checking the status of every credential at every access request introduces a significant additional computation overhead.

In ÆTHER we have chosen to adopt the first mechanism of short expiration time periods for the issued credentials. We have rejected the use of CRLs since their correct implementation and management is particularly complex, especially in environments with a lot of participating entities. Online credential status mechanisms require the issuer to always be accessible by every verifier, a requirement that is not compatible with the disconnected nature of pervasive computing. In addition, the involvement of the credential issuer in every access control decision makes its use very expensive from a computation point of view. ÆTHER attribute credentials are issued with short expiration time periods over a location-limited channel established between the issuer and the subject. The issuer is able to specify in the *renewable* field if the credential can be renewed automatically before it expires or not. If the field is set, then the subject is free to contact the issuer any time before the expiration of the credential and request a renewal. This process takes place over a wireless transmission and requires the subject to be within the communication range of the issuer. If the *renewable* field is not set, or if the issuer cannot be contacted the credential expires. If the subject needs another certificate of the same type then a location-limited channel with the issuer must be established again.

The main problem with the refreshing mechanism is the choice of the expiration time period. If it is too short then the issuer becomes frequently involved in renewing procedures; if it is too long the possibility of having credentials that have to be revoked but cannot increases. The ideal design choice would be to let the end user decide the validity period of the credentials she issues. However, this is an unrealistic approach as it assumes that every user is able to analyze all the risks involved in such a decision. In our current design the typical validity period we suggest is one hour. We believe that this choice satisfies the above requirements, however we must note that this is just a suggestion. The optimal validity period depends on the exact application scenario, requires extensive research and analysis, and is therefore outside the scope of this dissertation.

4.3 ÆTHER$_0$

In this section we present the first instantiation of the general ÆTHER model, namely ÆTHER$_0$, which has been designed to address the authorization needs of small pervasive

computing domains whose management requirements are simple. We consider the management requirements of a domain simple when the number of users that have owner rights on the participating devices and the number of the devices themselves are relatively small. For example a small household with one or two members can be such a domain, or the domain of a Personal Area Network (PAN) consisting of all the devices that a single user carries on himself. Furthermore, ÆTHER$_0$ utilizes only symmetric cryptography in order to provide security services. Consequently it is appropriate for devices that have particularly limited processing capabilities, like for example simple sensors.

4.3.1 Management Model

In ÆTHER$_0$ we have two classes of devices. The first one is comprised of normal pervasive computing devices that participate in the environment and offer services that have to be protected from unauthorized access. These devices can be simple without extensive processing power or large amounts of available memory. Examples of such devices include printers, switches that control various functionalities like doors and lamps, television sets, different types of sensors, and audio speakers. The second class consists of devices that directly represent a user in the management model. These are equipped with adequate physical interaction interfaces, such as keypads and screens, as well as more advanced processing capabilities and memory capacities than the devices of the previous class. Suitable devices for this purpose can be PDAs, mobile phones, or more traditional mobile computing devices such as laptops and tablet PCs. Users use these devices to store cryptographic data that can be used for authentication purposes as well as policy data that are used for authorization. In the ÆTHER$_0$ instantiation we call these devices *alpha-pads*. Each and every device that participates in an ÆTHER$_0$ PAD is uniquely identified by a hexadecimal string. This is constructed by concatenating the hardware addresses of the location-limited channel interface and the hardware address of the normal networking interface with a randomly generated number. The resulting value is given as input to a hash function and the output is encoded to hexadecimal format. This string is stored in the device itself, is public and is used to identify the device that generated it in the utilized policy statements.

Although ÆTHER$_0$ does not depend on any external centralized infrastructure for its

operation, it follows a locally centralized management model. Before a pervasive device of the first class can participate in a domain and offers its services it must be bound by the owner's alpha-pad. The binding takes place over a location-limited channel established between the two devices with the help of the owner and is enforced through a shared secret. The alpha-pad device generates a symmetric secret key (included in a binding policy statement) and sends it to the new device over the location-limited channel. Now the new device is bound by the owner's alpha-pad and can authenticate it through this shared secret. A user is free to have more than one alpha-pad and a slave device can be bound to more than one of them as well. However, a binding procedure can only take place over a location-limited channel and only if a user has expressed her intent to do so via some physical method (like pressing a button) on the device to be bound. The owner must repeat the same process for all the devices of the first class she owns that participate in the domain and have services to offer. Hence, at the end of this process all devices are bound to one or more alpha-pads and share secret keys with the alpha-pad that was used to bind them.

Using the alpha-pad the owner can now create further policy statements for the bound devices. These policy statements are sent to the bound devices over a wireless transmission medium and are authenticated, as well as encrypted, using the previously established shared secrets. The issuing alpha-pad as well as the subject device are identified in the policies with the unique hexadecimal strings. Each alpha-pad in ÆTHER$_0$ has its own localized namespace and is responsible to bind and define authorization policies for normal devices that offer services. Each bound device follows the namespace and participates in the domain defined by its alpha-pad. Therefore, a Pervasive Authority Domain (PAD) in ÆTHER$_0$ is represented by a specific alpha-pad that has been used to bind and configure a set of ordinary devices.

When a user wishes to access a service provided by a device she owns, she can simply use the alpha-pad that has been used to bind the target device[4]. The target identifies the alpha-pad through the binding shared secret key and performs the requested operation. If a user wants to access a service of a device that belongs to another user, then both users

[4]It is also possible to use a non-alpha-pad device she owns, subsection 4.3.5 presents all the relevant protocols in detail.

need to be at the same physical space. When the two users meet we assume that each one carries his alpha-pad device on him. Following our general user-based trust model, for a user to allow another user to use one of his devices that offers some services, he has to state his intent that the other user is trusted to do so through some physical action with his alpha-pad device. To express this trust the two users establish a location-limited channel with their alpha-pads and agree on a symmetric secret key over it. We call this procedure *mutual binding*. Furthermore, each user assigns authority attributes to the other user. After this procedure both alpha-pads are authenticated through the established shared secret and authorized with the assigned authority attributes.

A user can now use her alpha-pad (or another device she owns) to access a service provided by a device of the user she performed the mutual binding with. The requesting user's device securely contacts its alpha-pad (this step is obviously not performed if the requesting user directly uses his alpha-pad) and states the request to access the target device. The source device also generates and sends a new secret key. The alpha-pad finds the mutual binding policy statement with the target's alpha-pad and forwards the access request and the new key to it protected with their mutual binding key. The target alpha-pad transmits everything to the target device encrypted using their shared binding key. Additionally, the target alpha-pad transmits to the target device the authority attributes it has assigned to the requesting alpha-pad during their mutual binding process. The target device based on the authorization policies it locally has and the authority attribute assignments it has been forwarded can now reach an access control decision for the requested action. Fig. 4.5 summarizes this process (dashed lines represent wireless transmissions, while continuous lines represent location-limited channels).

Example. Alice visits Bob's home. She carries her mobile phone which plays the role of her alpha-pad and a pervasive computing wrist watch (which is a normal device that has been previously bound to her mobile phone). Bob has a PDA as his alpha-pad and a remotely controlled lamp. When the two users meet they mutually bind their alpha-pads, establishing a shared secret key, and they assign authority attributes to each other. We assume that Bob assigns to Alice the authority attribute *(group, visitor)*. These assignments stay with the issuer (Bob in this case); they are not quantities that are given

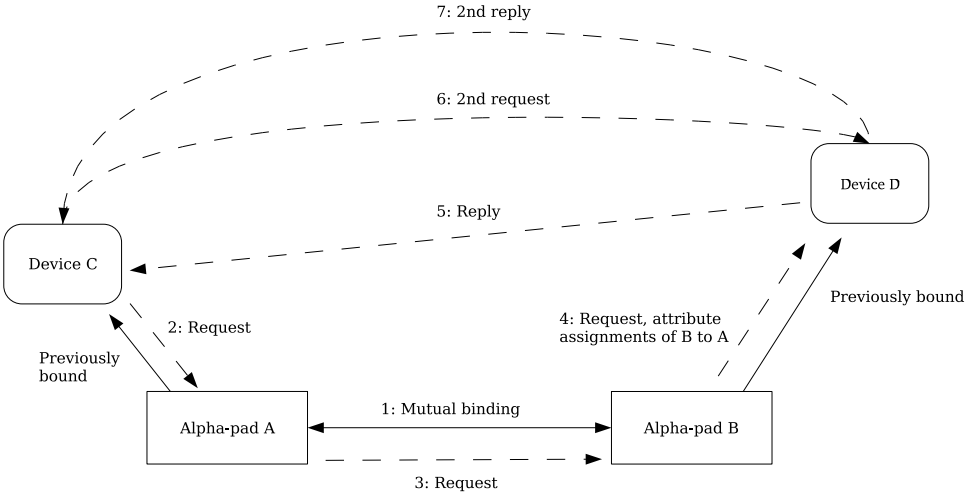

Figure 4.5: Overview of the ÆTHER$_0$ management model.

to the subject (Alice). Moreover, Bob has sometime ago bound his lamp to his PDA and issued a positive authorization statement to it allowing principals with the authority attribute *(group, visitor)* to control it. At some point Alice wants to switch the lamp on using her wrist watch. The wrist watch forwards the access request and a newly generated secret key to Alice's mobile phone encrypted with their shared key. The mobile phone decrypts the request, finds the mutual binding with Bob's PDA and the key they have established, encrypts the request and the new key with it, and in turn forwards them to Bob's PDA. The PDA decrypts the request and identifies that the target of the request is the lamp. It then sends the *(group, visitor)* attribute assignment that Bob has given to Alice to the lamp along with the access request and the new secret key. The subject identifier in the attribute credential that is sent to the lamp is the one of the device that initiated the request, in this case Alice's wrist watch. This message is protected with the secret key that the lamp and the PDA share. All the messages are encrypted and authenticated using the respective previously established secret keys. The lamp can now authorize the request based on the authority attribute assignments it has been forwarded and its local policy. The lamp transmits its authorization decision to the wrist watch. The communication with Alice's wrist watch is protected with the new key. Further access requests from the wrist watch can be issued directly to the lamp (and vice versa), until

the new shared key expires and the lamp deletes it from its policy database.

4.3.2 Policy Statements

ÆTHER$_0$ uses all the statements we have already presented in subsection 4.2.4. However, it adds a new field to the binding policy statement:

```
aether version: 0
type: binding
issuer: <Quoted hexadecimal string>
subject: <Quoted hexadecimal string>
shared secret: <Quoted hexadecimal string>
```

The *shared secret* field is used in order to store the shared secret key that is generated by an alpha-pad that binds another device. The identifier of the alpha-pad is kept in the *issuer* field and the identifier of the bound device in the *subject* field. Moreover, ÆTHER$_0$ introduces a new policy statement, the mutual binding policy statement:

```
aether version: 0
type: mutual binding
issuer: <Quoted hexadecimal string>
subject: <Quoted hexadecimal string>
not valid before: <Year/month/day-hour:second>
not valid after: <Year/month/day-hour:second>
shared secret: <Quoted hexadecimal string>
```

When two alpha-pad devices are mutually bound they agree on a shared symmetric secret key. In addition, each one issues a mutual binding policy statement to the other. Obviously these two statements have the same *shared secret* field that contains the agreed key and the values of the fields *issuer* and *subject* interchanged.

4.3.3 Policy Distribution

Binding policies are issued by alpha-pads to service providing devices over a location-limited channel. The owner of the devices brings them together and establishes the channel by specifying on the device to be bound his intent. The device (B) generates and sends a binding request message ($BREQ$) that contains its identifier (ID_B). The alpha-pad (device A) receives the subject's identifier, generates a new symmetric secret key (S_{AB} of

256 bits size), encodes it in hexadecimal format and builds the binding policy statement (BP_{AB}[5]). This is transmitted to the device to be bound (B). The owner also selects on the alpha-pad the authority attributes that he wishes to assign to the bound device. These assignments stay locally in the database of the alpha-pad. After this process the new device is considered bound by the specific alpha-pad (see Fig. 4.6).

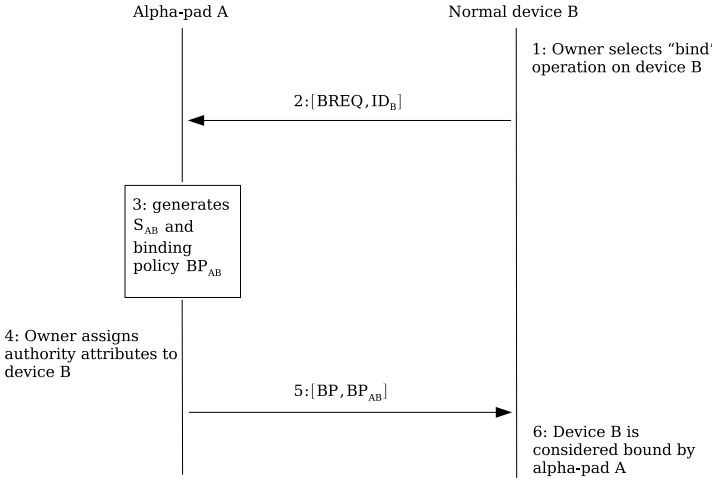

Figure 4.6: Binding of device B by alpha-pad A over a location-limited channel.

The generated binding policy BP_{AB} of the above example could be the following (please note that the hexadecimal strings are not presented in their entirety for readability reasons):

aether version: 0
type: binding
issuer: "ab3ddca94cef74c5fe2e4e7330bb5a1a5c14569b2eb" (Identifier of A, i.e. ID_A)
subject: "e2d53afadf7c29899510821f6499c0a4a509bab9afd" (Identifier of B, i.e. ID_B)
shared secret: "67cc50c5bec56ab895772c2ea2ed74f80d30ee9" (S_{AB})

Once a device is bound the owner can use the alpha-pad that bound it in order to issue more policies for it, like for example positive authorization statements for the services it has to offer. These policies are sent to the new device over a normal wireless interface. They are authenticated, encrypted and integrity protected using the 256-bit secret key that the alpha-pad and the target device now share. The authenticity and the integrity of the messages that contain the new policies are guaranteed with the use of a Hashed

[5]Subscript is <*issuer, subject*>.

Message Authentication Code (HMAC)[6]. The confidentiality of the messages is protected with a symmetric encryption algorithm (we use AES in Cipher Block Chaining (CBC) mode [DR02]).

Mutual binding policy statements are issued by two alpha-pad devices to one another when their corresponding users bring them together and establish a location-limited channel. These statements contain the agreed shared secret between the two devices. Moreover, each user selects the authority attributes that are assigned to the other party. This process also takes place over the location-limited channel. Fig. 4.7 presents the protocol.

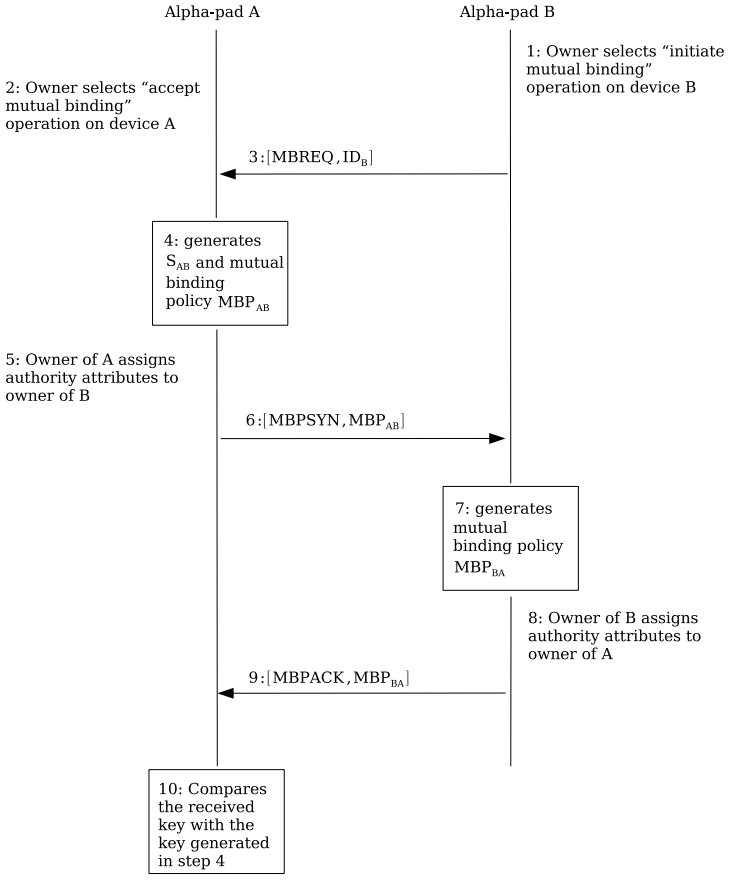

Figure 4.7: Mutual binding of alpha-pads A and B over a location-limited channel.

After both alpha-pad owners have expressed their intent to establish a location-limited channel for creating a mutual binding, device B sends a mutual binding request message

[6]Our design is modular and can support different HMAC algorithms, like HMAC-SHA-1, HMAC-SHA-256 or HMAC-MD5 [KBC97].

($MBREQ$) to device A along with its identifier ID_B. Device A generates a new 256-bit symmetric key S_{AB}, builds the mutual binding policy statement with itself as the issuer and device B as the subject (MBP_{AB}), and requests from its owner to select the authority attributes to be assigned to the owner of device B. These assignments are stored in the policy database of A. The policy (MBP_{AB}) is sent to B which reads the key from MBP_{AB} and builds its own version of the policy, MBP_{BA}, which is the same as the one received apart from having the values of the *issuer* and *subject* fields interchanged. Then the owner of B selects the authority attributes to be assigned to the owner of A and sends MBP_{BA} to A. A compares the *shared secret* fields of MBP_{AB} and MBP_{BA} and if they are the same (i.e. no corruption has occurred) the two devices are considered mutually bound.

In both the normal binding protocol between an alpha-pad and an ordinary device, and the mutual binding protocol between two alpha-pads the authority attribute assignments can take place after the completion of the protocols. However, we have included them as specific steps in order to illustrate that they are logically associated with the respective processes.

4.3.4 Authority Revocation

As we have already discussed, the approach of the general ÆTHER architecture is to avoid explicit revocation mechanisms and to rely on short expiration time periods and refreshing mechanisms for the issued authorization policies. In the ÆTHER_0 instantiation of the general model the policies that expire are the mutual bindings and the authority attribute assignments. The normal bindings between the alpha-pads and the rest of the devices of a user do not expire. If the user wishes to revoke a binding between an alpha-pad and a service providing device she owns she has two alternatives. The first one is to use the hardware reset switch and delete all the policy data from the bound device, in essence returning its state to a new unbound device. The second alternative is to use the alpha-pad device used to create a binding with a target device and issue a *revoke binding* operation. Alpha-pads that have been used to bind normal devices can issue such a command to a previously bound device. The target device deletes the binding policy of the issuing alpha-pad, as well as all the other policy statements that have been issued by it (identified by

the alpha-pad's identifier in the *issuer* field and authenticated through the shared secret). If a revoke binding operation removes the final alpha-pad owner of a device its effect is obviously the same as the use of the hardware reset switch.

Mutual binding policy statements established between alpha-pads do not contain a *renewable* field since they can always be renewed. Before the mutual binding expires the two devices use their shared key to create a new mutual binding. This time the protocol takes place over the wireless channel, provided of course that both devices are within the communication range of each other. The authenticity, confidentiality and integrity of the renewal protocol is protected with the secret key that the devices share. If the devices are not able to contact each other then the mutual binding expires and is deleted along with all the associated authority attribute assignments. In case the owners wish to offer services to each other again they have to establish a new mutual binding over a location-limited channel. Attribute assignments also expire and are deleted by the issuer when the mutual binding with the corresponding subject alpha-pad expires. Whether they can be renewed or not depends on the value of their *renewable* field. If the field is set then when the mutual binding is renewed the assignments are renewed too. Otherwise they are deleted by the issuer device. In the case that normal devices have authority attribute assignments for other principals that they have been given to them by their alpha-pads (see the example in subsection 4.3.1) these are also renewed before they expire (or not based on their *renewable* field). If the field is set the device contacts its alpha-pad before the assignment expires and requests a renewal.

4.3.5 Service Discovery and Access

When a user visits a new pervasive computing environment she initially needs to find what services are offered so that can consequently issue access requests. In ÆTHER$_0$ alpha-pad devices are used to securely bind all service offering devices that belong to a particular owner. Furthermore, owners use their alpha-pads to issue authorization policies that control access to the services that they want their other devices to offer. Hence, alpha-pads are aware, and store in their local databases, what services are provided and by which devices. Two owners that establish a mutual binding between their alpha-pads

state with this action that they trust each other, not completely but up to the level implied by the authority attributes they assign to each other. The services that an owner has to offer can be securely advertised to the other party, using the secret key the two alpha-pads share after the performance of the mutual binding procedure. Service advertisements include the identifier of the provider, a service identifier ($Service_{ID}$) which is meaningful only to the provider and is used to locally distinguish the different provided services, and a textual description of the service. Fig. 4.8 illustrates this concept (continuous lines represent location-limited channels while dashed lines normal wireless transmissions). Similarly, when an alpha-pad is used to bind a normal device service advertisements and the identifiers of the devices that offer them can be released from the alpha-pad to the newly bound device. Each device maintains what we call a Current PAD View (CPV). The CPV is a list of the services and their associated details (the service's identifier, the service's textual description, the identifier of the provider, and the service's access interface if the device managed to get access to the service in question) that gets populated with entries when a device receives service advertisements.

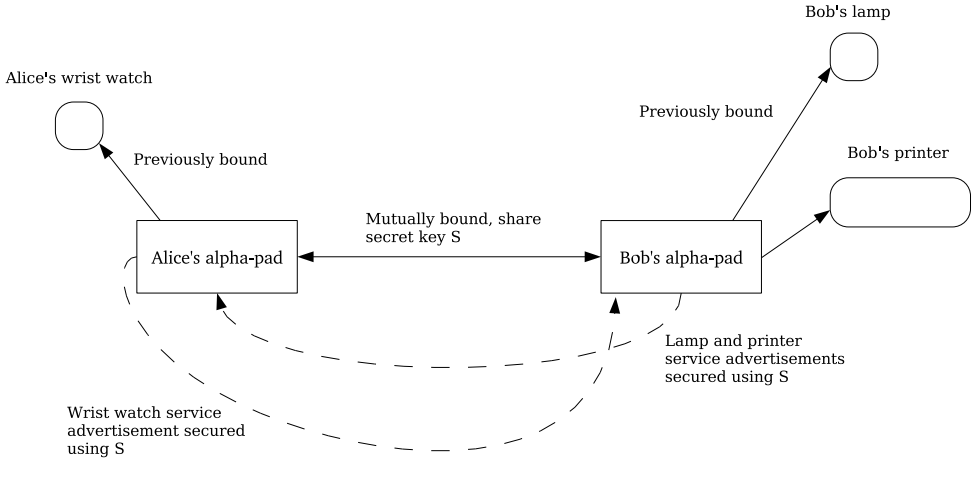

Figure 4.8: Service discovery in ÆTHER$_0$.

Services can also be provided by the alpha-pads themselves, for example a service sharing files that a user keeps stored on his mobile phone. Therefore, there are five different access protocols we need to address. The first one considers the case in which a normal device wants to access a service provided by the alpha-pad that bound it (and the opposite;

an access request from an alpha-pad to a normal device it has bound before). The second one when both the requester and the provider are normal devices bound by the same alpha-pad. The third one when an alpha-pad is used to access a service provided by another alpha-pad with which it shares a mutual binding. The fourth case involves an access request from a normal device to an alpha-pad with which the requesting device does not share a binding, but the alpha-pad of the requester shares a mutual binding with the target alpha-pad (and the opposite). The final protocol addresses an access request from a normal device to another normal device that have been bound by different alpha-pads that share a mutual binding. The following paragraphs analyze these protocols.

4.3.5.1 Normal Device to Alpha-pad

In this case the access request and reply protocol is trivial since the normal device and the alpha-pad share a secret key that has been previously established when the owner bound the device using the alpha-pad over a location-limited channel. The application of the HMAC is denoted as $a = H_{S_{AB}}(b)$, where b is the input to the HMAC algorithm H, S_{AB} the symmetric shared key that entities A and B share, and a the output. Fig. 4.9 presents this protocol. The opposite protocol is exactly the same and therefore not presented.

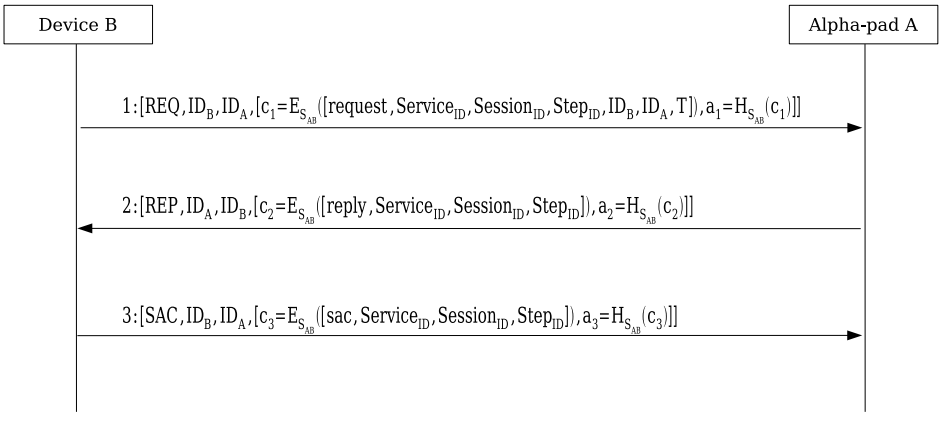

Figure 4.9: Normal device to its alpha-pad service access request protocol.

The initiator, device B, sends an access request to alpha-pad A, the service provider. The packet header contains a message identifier (REQ), the sender's ID in the clear in order to allow the receiver to locate the secret key with which the payload is protected, and

the receiver's ID. The payload contains the service request, a service identifier ($Service_{ID}$), a session identifier ($Session_{ID}$), a protocol step identifier ($Step_{ID}$), and the identifiers of A and B (ID_A and ID_B). Furthermore, a timestamp (T) is also included to protect against replay attacks. The payload is encrypted using S_{AB}, the key both devices share from when A bound B. Furthermore, message authentication is performed by computing an HMAC using c_1 (the ciphertext of the payload) and S_{AB}. Authentication and confidentiality is guaranteed for the remaining of the messages using the same way[7]. The alpha-pad passes the set of authority attributes it has previously assigned to B (i.e. when it bound it) and the authorization policies relevant to the requested operation to its ÆTHER$_0$ inference engine. The result is a boolean value that is transmitted to B in step 2 (for the sake of the example we assume that the inference engine reached a positive decision) along with the same $Session_{ID}$, an incremented $Step_{ID}$, and the access interface of the service. In the final step B transmits a service access message (with message identifier SAC) of the requested service to A.

4.3.5.2 Normal Device to Normal Device (Same Alpha-pad)

This protocol addresses the case where a normal device wishes to access a service provided by another normal device. The two devices do not share any kind of shared key or relationship, apart from the fact that both of them have been bound in the past by the same alpha-pad. Therefore, the alpha-pad shares a different secret key with each device. Any protocol that has appeared in the security literature for authentication and key exchange between two parties based on a mutually trusted third party can be used in this case. The two devices use such a protocol and with the help of the alpha-pad they establish a new shared secret. After the establishment of the new shared secret the two devices communicate directly using the protocol we have presented in 4.3.5.1.

The only addition is that the alpha-pad forwards to the device that offers the requested service the authority attribute assignments it has for the requesting device during the key exchange protocol. Now the provider gives as input to its inference engine the authority attribute assignments it has been forwarded, the local policy statements, the access request

[7]We use the *encrypt-then-authenticate* (ETA) mechanism of Krawczyk [Kra01].

and gets as an output whether the request is granted or denied. The newly established shared secret between the two devices does not remain active forever, but expires. The typical validity time period we suggest is a few hours. During this period the requester can directly contact the provider without going through the alpha-pad again. Fig. 4.10 presents the above process using the Wide-Mouth Frog protocol [BAN90], however note that any similar protocol, like for example the Otway-Rees protocol [OR87], can be used for the same purpose. We denote the set of authority attributes that A has assigned to B as AA_{AB}[8].

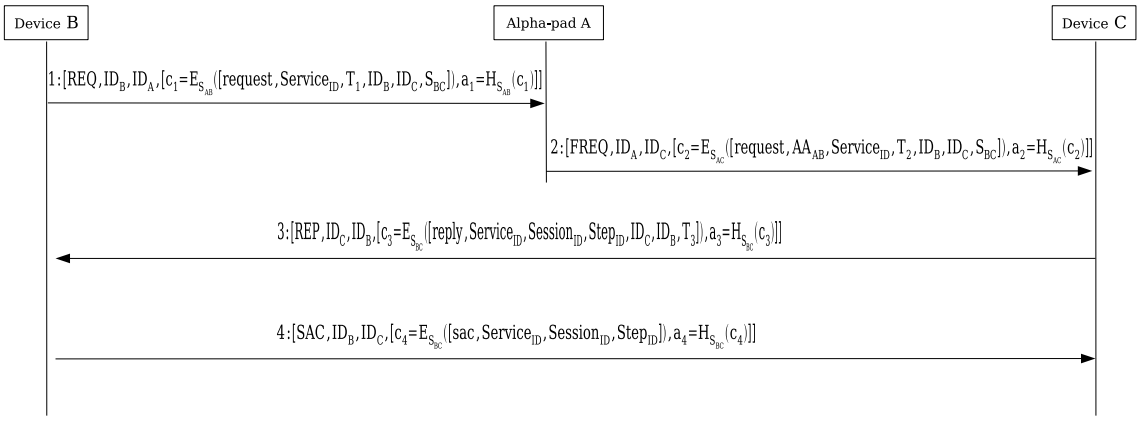

Figure 4.10: Normal device to normal device (same alpha-pad) protocol.

4.3.5.3 Alpha-pad to Alpha-pad

Since both alpha-pads share a secret key based on the mutual binding the owners have performed over a location-limited channel, this protocol is exactly the same as the protocol presented in 4.3.5.1.

4.3.5.4 Normal Device to Alpha-pad (Different Alpha-pad)

In this situation we have an access request initiated from a normal device (C), that has been bound in the past by alpha-pad A, targeted at a service provided by another alpha-pad, namely B. The assumption is that alpha-pads A and B share a mutual binding. Since C and A share a key (from when A bound C) and also A and B share a key as

[8]Subscript is $<issuer,\ subject>$.

well (from when they were mutually bound), alpha-pad A can play the role of a trusted intermediary between C and B. The protocol is similar to the one we have examined in 4.3.5.2. However, there is no transmission of authority attribute assignments; alpha-pad B has already assigned authority attributes to alpha-pad A when they were mutually bound, and therefore has all the required information to pass to its inference engine in order to get an authorization decision for C's request.

4.3.5.5 Normal Device to Normal Device (Different Alpha-pad)

The protocol we present in this paragraph covers the situation where an access request is made from a normal device (C) to another normal device (D). However, the two devices have been bound by different alpha-pads. Specifically, device C has been bound by alpha-pad A and device D by alpha-pad B. The assumption here is that these two alpha-pads share a mutual binding and that the owner of B has assigned authority attributes to the owner of A. Fig. 4.11 illustrates the protocol.

Device C sends the request to its alpha-pad protected with their shared key S_{AC}. The request includes a new symmetric secret key that C generated, namely S_{CD}, and a timestamp T_1 to protect against replay attacks. Alpha-pad A forwards the request and the new key to alpha-pad B. This message is protected with the mutual binding key A and B share. B identifies that the target of the request is D and forwards to it the new key along with the authority attributes (AA_{BA}) it has previously assigned to A. The credentials that are sent to D by B have as their *subject* field value the identifier of the requester, in this case ID_C. D decrypts this message with the key it shares with its alpha-pad B and passes to its inference engine the request, the set of authority assignments it has been forwarded, and its local policies. The result is sent directly to C protected with the new secret key (S_{CD}). If the result is positive the REP message also contains the access interface of the requested service. The new key S_{CD} can now be used for further communications between C and D (and vice versa). However, after S_{CD} expires the whole process has to be performed again.

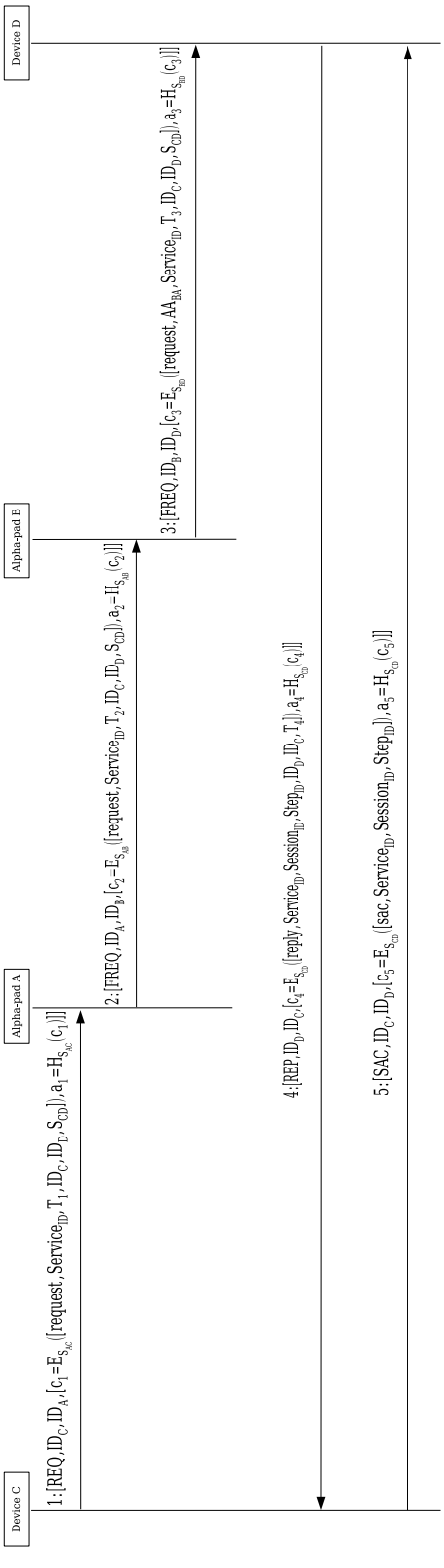

Figure 4.11: Normal device to normal device (different alpha-pad) protocol.

4.3.6 Inference Engine

The inference engine of ÆTHER$_0$ is invoked by the provider of a service after the requester has made an access request according to the protocols presented in the previous section. The algorithm used is the following (illustrated by the flowchart presented in Fig. 4.12):

1. When a request is received from a principal the default value of the access decision is set to *false*.

2. All the authorization policies relevant to the request are collected from the local policy database of the service provider.

3. If context attributes are used in the authorization policies the engine checks to see if any of these attributes have dynamic local context values. If there are context attributes with dynamic values used to define the authorization policies, their current values are retrieved from the provider's LCP. If no context attributes are required the execution jumps to step 5.

4. Based on the CAS statements the provider locally has it identifies the principals that are responsible for maintaining the membership of principals in the dynamic sets that represent the context attributes used in the identified policies. Then the membership status of the requester for the specific context attributes is requested from the relevant principal (e.g. a sensor device). This is a normal access request that is accomplished using the access protocols we have already presented. If there are negative authorizations for the requested operation and negative authorizations have precedence over positive authorizations the value of the access decision is set to *false* and returned. If the requester has the necessary context attributes and no negative authorizations exist (or they do not have precedence over positive ones) the decision value is set to *true*.

5. Then the engine checks to see if there are authority attributes required. If none are required the decision value is returned.

6. If there are authority attributes required, the engine searches the local policy database to find all the authority attributes that have been assigned to the requesting prin-

cipal. If there are negative authorizations for the requested operation and negative authorizations have precedence over positive authorizations the value of the access decision is set to *false* and returned. Otherwise the execution continues with the next step.

7. If the authority attributes of the requester satisfy the authorization policies the decision value is set to *true* and returned. Otherwise it is set to *false* and returned.

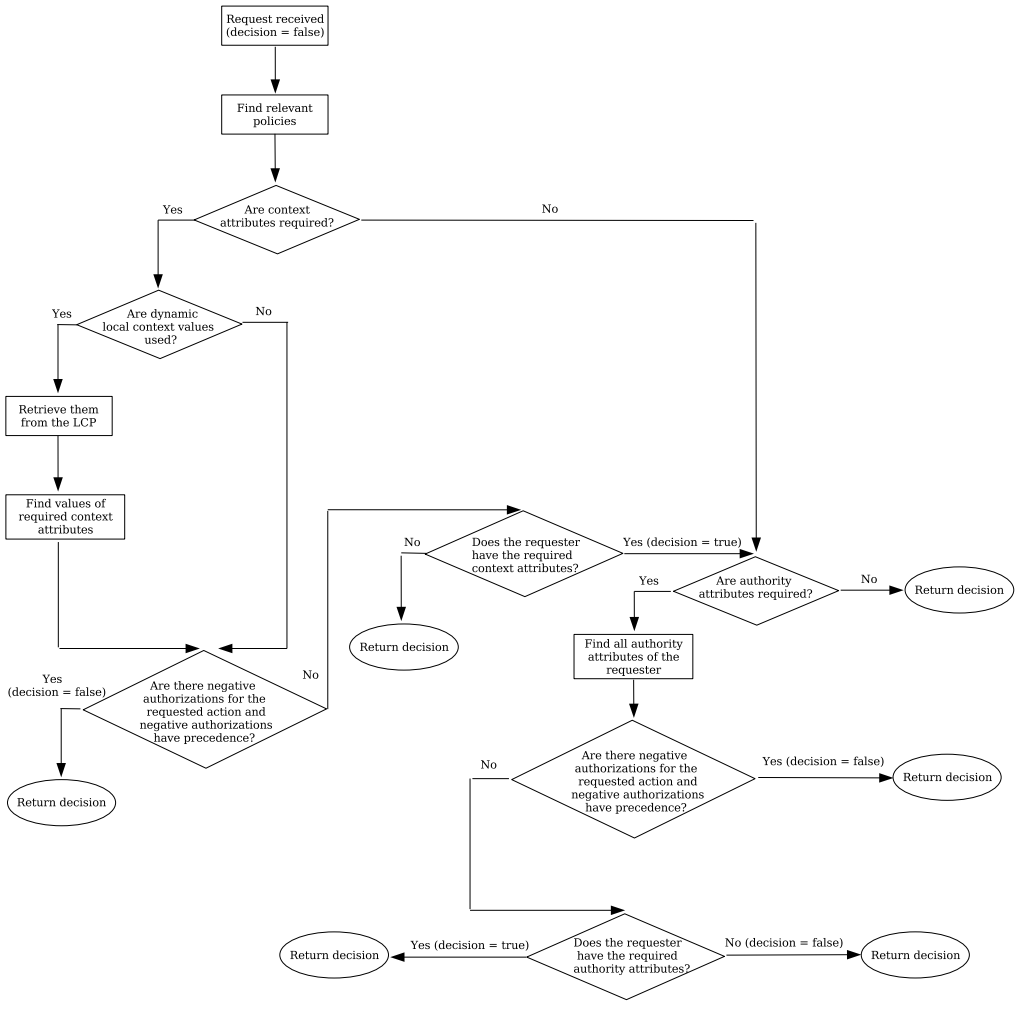

Figure 4.12: Access control decision flowchart of the ÆTHER$_0$ inference engine.

4.3.7 Security Considerations

The access request protocols of ÆTHER$_0$ we have presented in the previous sections are all based on the Wide-Mouth Frog protocol. Its basic assumption is that it requires loosely synchronized clocks between the participating entities in order to protect against replay attacks by utilizing timestamps. Other similar solutions, like for example the Otway-Rees [OR87] or the Needham-Schroeder [NS78] protocols, avoid the use of timestamps by using nonces and therefore more message exchanges to achieve the same goals. The only requirement of ÆTHER$_0$ regarding access protocols is the previous establishment of symmetric keys over location-limited channels according to the management model we have already presented. All access request protocols can be modified to be based on other similar symmetric authentication and key exchange protocols. Furthermore, the general ÆTHER model already makes the assumption of loosely synchronized clocks among the participants by using expiration time periods for authority revocation purposes. The main reason behind the selection of the Wide-Mouth Frog protocol is its simplicity and the small number of exchanged messages that it requires.

4.4 ÆTHER$_1$

This section analyzes the second instantiation of the general ÆTHER model, ÆTHER$_1$. The main goal of this instantiation is to support the authorization requirements of large pervasive computing domains that have multiple owners with complicated security relationships. Such environments consist of large numbers of devices whose ownership rights can be shared among many principals. Example domains include large households with four or more members, businesses and other shared working environments, and commercial stores with many customers. Furthermore in ÆTHER$_1$ we address the main disadvantage of ÆTHER$_0$, that is the reliance on alpha-pads as the locally centralized administrating devices. ÆTHER$_1$ extends traditional RBAC in order to allow the sets of entities that have authority over a specific authority attribute, or Authority Attribute Sets (AASs) according to the terminology of ÆTHER$_1$, to grow dynamically. This allows the delegation of authority over the ability to assign authority attributes to principals, removing the ne-

cessity of a user always having to carry a specific device (the alpha-pad in ÆTHER$_0$) with her in order to authorize other users she encounters. To achieve this ÆTHER$_1$ utilizes asymmetric cryptography and certification of authority attributes through digital signatures. Therefore, it is more fitting to PADs that consist of devices that have the required information processing capabilities to handle asymmetric cryptography algorithms.

4.4.1 Management Model

In ÆTHER$_1$ all participating devices have a built-in asymmetric cryptography key pair. According to the design approach that is advocated by SPKI/SDSI [CEE+01], we also assume that an entity is identified, and therefore named, by the public key of this key pair[9]. The corresponding private key is kept protected within the device. ÆTHER$_1$ allows every device to issue bindings and assign authority attributes to other devices by generating authority attribute certificates signed with their private key. Therefore, every device has its own local namespace and is allowed to define its own names for attributes. Public keys are encoded in hexadecimal format and are used to identify principals in the policy statements used in ÆTHER$_1$.

A Pervasive Authority Domain (PAD) in ÆTHER$_1$ is defined as the initial set of relationships between attributes and principals specified in a security policy and is a logical representation of a ubiquitous computing environment. The owner of several devices creates a PAD by specifying in policies which principals are trusted to certify which authorization and context attributes. Moreover, the owner creates policy entries for controlling what authorization and context attributes a principal must possess in order to get specific access rights to a resource provided by a device. These policies are distributed to subject principals by issuers over wireless transmissions. In order to bootstrap security relationships and authenticate the distribution of these policies we again rely on location-limited channels for creating bindings between principals. Binding policies in ÆTHER$_1$ serve the purpose of binding the subject principal to the issuing principal. This is similar to the imprinting functionality of the Resurrecting Duckling system, but instead of sharing a common secret (as we do in ÆTHER$_0$) we utilize a signed statement to achieve the same goal. The

[9]The same association between entities' names and public keys can easily be satisfied with the use of an Identity-Based Encryption (IBE) system, like for example the one presented in [BF01].

embedding of the binding policy to the subject device must take place over a location-limited channel in order to ensure its authenticity. After a device has been bound, it can accept new policy entries remotely by the principal that bound it. These new policies are fully authenticated since the subject has already been bound by the issuer and knows its public key. As in ÆTHER$_0$, an ÆTHER$_1$ device can be bound to more than one devices.

When a principal wishes to issue an attribute certificate for another principal the two devices establish a location-limited channel with the help of their human owners. Although any principal is allowed to issue a certificate for any authority attribute, only the ones that are issued by members of the AAS responsible for the specific authority attribute are considered valid. An Authority Attribute Set (AAS) is a policy statement that identifies the principals that are the sources of authority for a specific attribute. Membership in an AAS means that the corresponding principal is trusted to issue attribute credentials of the specific type. A member of an AAS is defined as a principal that has been explicitly denoted in the signed AAS statement as such, or an unknown principal that has been given the required attribute certificates (ACs) by at least a threshold number of members of the set. This threshold number is called Membership Threshold Value (MTV) and the accepted values are integers greater or equal to 2. This is in essence a delegation of the authority specified in the policy statement. The allowed depth of delegation is controlled by a *delegation depth* entry in the AAS definition statement that in essence implements integer delegation control. Although there have been arguments in the literature against integer control regarding the inability to predict the proper depth of delegation [EFL+99], the application domain of ÆTHER$_1$ makes its use particularly attractive. The owner of a PAD can use representative values for integer delegation control when she defines the AASs of the domain according to the importance of the attributes they authorize. Important attributes that convey a high value of trust can have small delegation depth values while more general attributes that authorize less important actions can have bigger values.

Example. In a domestic ubiquitous computing scenario the AAS policy statement for the attribute *(group, visitor)* that authorizes visitors to the house could specify a delegation depth and a membership threshold value of 2. This would allow a trusted visitor to introduce another visitor in the house, and in order for a visitor to be able to

do so she must be trusted by two existing members of the AAS. This AAS with principals Key_0 and Key_1 as the initial sources of authority is illustrated in Fig. 4.13. At some point the user of the device identified by key Key_0 establishes a location-limited channel with the user of the device identified by key Key_2. Key_0 issues an attribute certificate for the attribute *(group, visitor)* to Key_2 over this channel. Since the membership threshold value of the corresponding AAS statement has been defined as 2, ACs for the attribute *(group, visitor)* issued by Key_2 at this point are not valid. Later on Key_1 also establishes a location-limited channel with Key_2 and issues a *(group, visitor)* AC to it. Now Key_2 has the required number of certificates and dynamically joins the *(group, visitor)* AAS. It is now able to issue valid certificates for authority attributes of this type. Key_2 does so by establishing a location-limited channel with Key_3. This AC is valid and Key_3 can use it to support access requests.

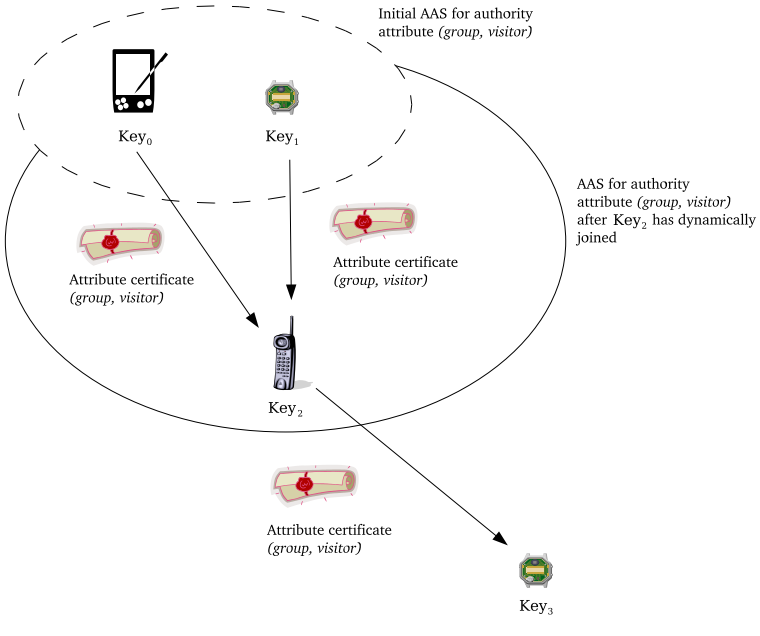

Figure 4.13: A dynamic Authority Attribute Set (AAS).

The ÆTHER$_1$ management model does not have semantic categories of devices based on their rights to issue attribute assignments or to create bindings as is the case in ÆTHER$_0$. As we have already explained, every device in ÆTHER$_1$ is free to issue authority attribute assignments and also to bind other devices to itself. However, some devices may not have

the required physical interaction interfaces to do so. As an example, a PDA can easily be used to bind other devices since it has a touch screen and/or a keypad to interact with the user who can use these facilities to express this intent. On the other hand, a small sensor with a diameter of a few centimeters lacks the physical interface capabilities to be used for this purpose. Such devices are bound by other devices that have the required interfaces over location-limited channels and are then issued policies over wireless transmissions. We assume that they have some physical way (like a small switch or a button) to allow the owner to indicate her wish to initiate a location-limited channel and bind them to some other device.

4.4.2 Policy Statements

ÆTHER$_1$ uses the complete set of policy statements we have analyzed in paragraph 4.2.4. However, in order to allow the authentication of these statements when they are transmitted over wireless media and not over location-limited channels (whose authenticity is guaranteed from the human users responsible for establishing them) a *signature* field is added to all of them. The *signature* field contains the digital signature, encoded in hexadecimal format, of the principal identified in the *issuer* field of the statement. It is calculated using an asymmetric cryptography algorithm over the complete text of the statement, starting from the first field up to but not including the *signature* field. Therefore, it is always the last field of a signed ÆTHER$_1$ statement.

A new policy statement that is introduced by the ÆTHER$_1$ instantiation is the Authority Attribute Set (AAS) statement:

aether version: 1
type: authority attribute set
issuer: <Quoted hexadecimal string>
attribute name: <String>
attribute value: <String, integer, or float>
membership threshold value: <Integer>
delegation depth: <Integer>
sources of authority: <List of comma separated and quoted hexadecimal strings>

AAS statements define the initial sources of authority for a specific authority attribute (the one identified with the *attribute name* and *attribute value* fields). These principals

are specified in the *sources of authority* field, identified by their public keys encoded in hexadecimal format. As we have seen, this set of principals can grow dynamically. A *delegation depth* field of -1 means that delegation of the specified authority is not allowed, a value of 0 means that delegation is allowed with no restrictions on the depth and any integer greater or equal to 1 defines the allowed depth. An AAS that has a *delegation depth* field of -1 and a *membership threshold value* field of 0 is defined as *static*, meaning that the set of principals that act as sources of authority for the corresponding attribute is not allowed to grow dynamically. Dynamic AASs are in essence dynamic sets as we have already described them in paragraph 4.2.4, while static AASs are traditional sets.

Another new type of policy statement in ÆTHER$_1$ is the Attribute Mapping Certificate (AMC). AMCs are used for secure interdomain communications and are fully explained in paragraph 4.4.7,

aether version: 1
type: attribute mapping certificate
issuer: <Quoted hexadecimal string>
attribute name: <String>
attribute value: <String, integer, or float>
subject attribute name: <String>
subject attribute value: <String, integer, or float>
subject sources of authority: <List of comma separated and quoted hexadecimal strings>
not valid before: <Year/month/day-hour:second>
not valid after: <Year/month/day-hour:second>

4.4.3 Policy Distribution

When an owner wishes to bind one of his devices to another device he owns he establishes a location-limited channel between them. On the device to be bound the owner selects the "bind" operation. This can be done either through a Graphical User Interface (GUI) or through some physical interface like a switch or a button. The target device (B) sends a binding request message ($BREQ$) to device A. The message includes B's public key encoded in hexadecimal format and signed using B's private key (K_B^{-1}). Device A receives this message, verifies the signature, builds binding policy BP_{AB} and sends it to B. B verifies the signature and inserts BP_{AB} to its policy database and A to its list of owners. At this point device B is considered bound to device A. A specific device can have more

than one owners, therefore the same process can take place again for device B. The binding policy BP_{AB} in this example (illustrated in Fig. 4.14) could be the following:

aether version: 1
type: binding
issuer: "3048024100cd462d1d7cf0aa50431616cb61ee55cafc83" (A's public key, i.e. K_A)
subject: "3048024100d1d645e46841f0b708b75dd95851d71819" (B's public key, i.e. K_B)
signature: "8314cea46400a27458b7f3eb6405dade9" (Signature with A's private key, i.e. K_A^{-1})

After A has bound B it can issue more policies to it over a normal wireless transmission. This communication between them is easily protected since now both devices know the public key of each other, and are sure about their authenticity since the exchange took place over a location-limited channel.

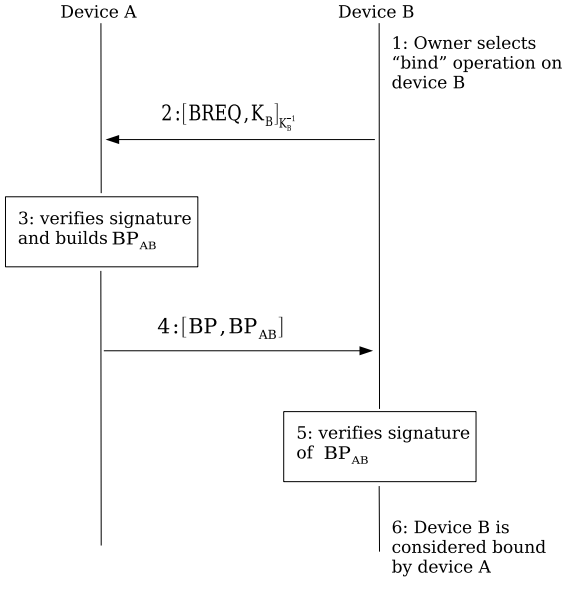

Figure 4.14: Binding of device B by device A over a location-limited channel in ÆTHER$_1$.

Authority attribute assignments (attribute certificates in the terminology of ÆTHER$_1$) are also issued over location-limited channels. We call this the ÆTHER$_1$ Attribute Certificate Acquisition Protocol (ACAP). The owners of the issuer and the subject devices establish this channel through some physical action that is integrated gracefully into their normal workflow. The issuer selects on her device (A) that she wishes to issue an attribute certificate and specifies its name, its value and if it is renewable. Device A generates

a random number R and sends this to device B over the location-limited channel in an attribute certificate issuing synchronization message ($ACSYN$). Device B receives this, signs the random number R with its private key (K_B^{-1}), and sends to A an acknowledgment message ($ACACK$) that also contains its public key (K_B). A verifies the signature and if the verification operation succeeds it builds the attribute certificate (AC_{AB}), signs it, and sends it to B (message $ACCERT$). B verifies the signature on AC_{AB} and inserts the certificate into its certificate database if the verification is successful. Fig. 4.15 presents this protocol.

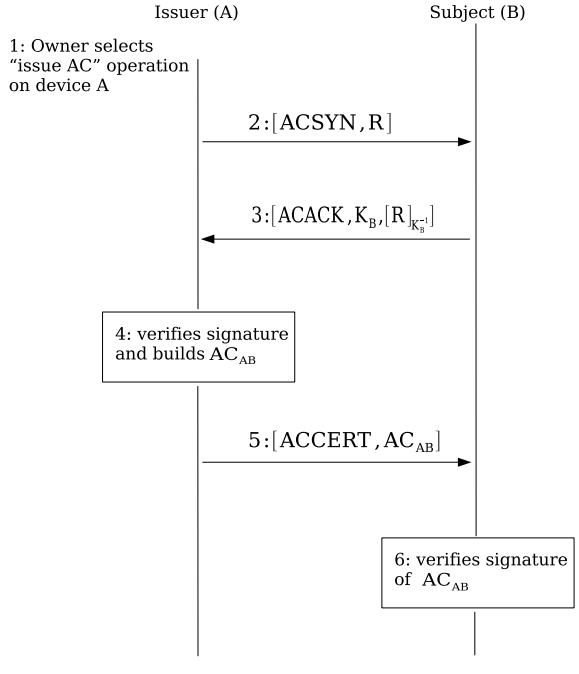

Figure 4.15: Attribute certificate acquisition protocol in ÆTHER$_1$.

As an example, if the owner of A has selected that the issued AC is for authority attribute *(group, visitor)* and that it is allowed to be remotely renewed, AC_{AB} would be the following:

aether version: 1
type: attribute assignment
issuer: "3048024100cd462d1d7cf0aa50431616cb61ee55caf" (A's public key, i.e. K_A)
subject: "3048024100d1d645e46841f0b708b75dd95851d71819" (B's public key, i.e. K_B)
attribute name: group
attribute value: visitor

not valid before: 2005/04/02-12:21
not valid after: 2005/04/02-13:21
renewable: 1
signature: "6f858cc2d0b548ff9382fe772adda6b0d81" (Signature with A's private key, i.e. K_A^{-1})

This certificate can be used by B to support access requests. The specific protocols for this purpose are presented in paragraph 4.4.5.

4.4.4 Authority Revocation

ÆTHER$_1$ follows the general ÆTHER revocation approach and relies on short expiration time periods and refreshing mechanisms for the issued credentials. When a user issues an attribute certificate the expiration time period can be defined. As we have already discussed the default period we suggest and use is one hour. Moreover, the user can specify whether the specific attribute certificate can be renewed remotely or not. The default behavior that ÆTHER$_1$ follows is to allow the remote renewal of issued certificates. At any time before the certificate expires the subject principal can contact the issuer principal over a wireless communication medium and request a renewal of the previously issued certificate. The exchanged messages towards this goal are fully secured since both principals know each others' public keys and are sure about their authenticity since the certificate was issued over a location-limited channel. If a certificate expires then it cannot be renewed. The issuer and the subject principals have to again establish a location-limited channel with the help of their human owners and go through the attribute certificate acquisition protocol we have presented in the previous paragraph.

Bindings between devices do not expire. In order to revoke a binding a user either uses the hardware reset switch on the bound device or uses a device that has issued a binding and sends a revoke binding command to the subject device. In the first case all the policies from the bound device are completely deleted. In the second case, the revoke binding command can only be invoked by an issuer device that has previously bound the specific subject device. The command is authenticated through a digital signature generated with the private key that corresponds to the public key the issuer used to create the binding with the subject and is transmitted over a normal wireless channel. Subsequently the subject deletes all the policies that the revoked issuer has issued to it in the past.

4.4.5 Service Discovery and Access

In ÆTHER$_1$ the discovery of available services is performed according to two different methods, namely the *eager* and the *lazy* service discovery and access protocols. As in ÆTHER$_0$, every ÆTHER$_1$ device maintains a list of all the available provided services it knows about. We call this list the Current PAD View (CPV) and its maintenance strategy depends on the utilized service discovery method. The following paragraphs present these methods and analyze the access protocols.

A concept that we use in both strategies is that of the Reduced Access Control List (RACL). The RACL is constructed by the service provider and is sent to a requester in order to aid the latter in identifying the authority attribute credentials that must be released to support a request. Since the general ÆTHER model supports negative authorizations, a provider cannot simply release the normal ACL (as represented by the *requires* field of the relevant policy statement) of a service to a requester. If this was the case, the requester could simply avoid releasing to the provider its credentials that subtract privileges. Therefore, the provider does not release the ACL itself, but a *reduced* ACL that is constructed from the ACL. In the RACL the provider includes just the names of the authority attributes that appear in the ACL without their values or the predicates that apply on them. As an example consider the following negative authorization:

```
aether version: 1
type: negative authorization
issuer: "3048024100cd462d1d7cf0aa50431616cb61ee55cafc83"
resource: television
operation: change_channel
requires: (@group == family_child) || (@age < 18);
signature: "4096816a21379e8d20d0206ff061f3af6a111dc0850"
```

The corresponding RACL is:

```
@group, @age
```

Note that context attributes are not included in the RACL. The concept of the RACL both helps a requester to identify the credentials it must release to support a request and it protects the provider from making public the exact ACL of a service. Of course, given a sufficiently simple *requires* field and a committed attacker the exact predicates can

be discovered through successive trials. To protect against this a provider can blacklist a requester that makes a certain number of successive failed attempts within a certain amount of time.

4.4.5.1 Eager

The main goal of the eager service discovery and access strategy is to protect the service providers from disclosing their advertisements to unauthorized parties. One of the main threats to service discovery is the enumeration of all available services by attackers. This can directly lead to the selection of vulnerable services to attack and constitutes an invasion of privacy. In order to secure the service discovery process, the eager strategy utilizes Public Resource Advertisements (PRAs). A PRA is a message which simply states that a service is provided. Specifically, a PRA contains the public key of the service provider (K_P) and the service identifier ($Service_{ID}$) which is a number used only by the service provider to distinguish between the different services it provides. PRAs are periodically broadcasted over a wireless communication channel every t seconds by providers for each service they have to offer[10]. When a device (the requester in our example with public key K_R) receives a PRA it checks to see if it already managed to get access to the service with the corresponding $Service_{ID}$. If this is the case then the PRA is ignored. Otherwise, it tries to access the provided service and enter it into its CPV. This is performed for every PRA a device receives.

The requester sends an access request to the service provider. The initial request message contains a packet identifier (REQ), the received service identifier ($Service_{ID}$), a session identifier ($Session_{ID}$), a protocol step identifier ($Step_{ID}$) and the service request. Furthermore, the public key of the requester (K_R), a digitally signed tuple containing the public keys of the two entities, a generated symmetric key (S_{RP}) and a timestamp (T) are also included. The entire message (except the packet identifier) is encrypted with the public key of the provider (K_P). Our key agreement protocol is based on [Aba02]. The provider replies with the RACL for the requested service (assuming that such a service is indeed provided), the same $Session_{ID}$ and an incremented $Step_{ID}$. The message is

[10]The exact value of t depends on the application scenario. The typical value we suggest (and use) is 60 seconds.

encrypted using S_{RP}, the negotiated session key. Furthermore, message authentication is performed by computing an HMAC using c_1 (the ciphertext of the message) and S_{RP}. Based on the received RACL, the requester builds a reply with a set of authority attribute credentials it holds that can support its request and sends it to the provider. When the complete set of credentials is sent the requester also sends a $CERTFIN$ message to denote that it has sent all the relevant credentials it holds. The service provider passes the set of credentials and the relevant policy statements it locally has to its ÆTHER$_1$ inference engine. The result is a boolean value that is transmitted to the requester in step 6 (for the sake of the example we assume that the inference engine reached a positive decision) along with the Detailed Resource Advertisement (DRA) and the access interface of the requested service. A DRA contains a textual description of the related service. At the end of step 6 we consider the ÆTHER$_1$ handshake to be over. When the requester receives the DRA it enters the information it contains into its CPV. Therefore, in the eager strategy when an owner lists the entries of her CPV she sees all the services in the current PAD that she is allowed to access based on her authority and current context attributes. When the owner decides to use a particular service we have the final step of the protocol where the requester constructs and sends a SAC message based on the received access interface and the session ends. At this point the provider deletes the symmetric key S_{RP}. An established session also expires and the corresponding session key is deleted after a certain amount of time has elapsed when no SAC message is sent from the requester to the provider. In our current design we have set the session expiration time period to 120 seconds. The protocol is illustrated in Fig. 4.16.

In order to enhance the performance of the eager access protocol we use an *authorization cache*. At the end of step 5 when the provider reaches an authorization decision for a request from a specific principal with a specific set of credentials it caches the result. When the same requester wishes to access a service that already is in its CPV it simply sends a new request for it. This request is the same as step 2 of the above protocol. The requester generates a new secret key (S'_{RP}), a new session identifier ($Session'_{ID}$) and a new timestamp (T'). The provider checks its cache to see if the specific requester has been successfully authorized in the past for the specific request. If it has then it replies with a

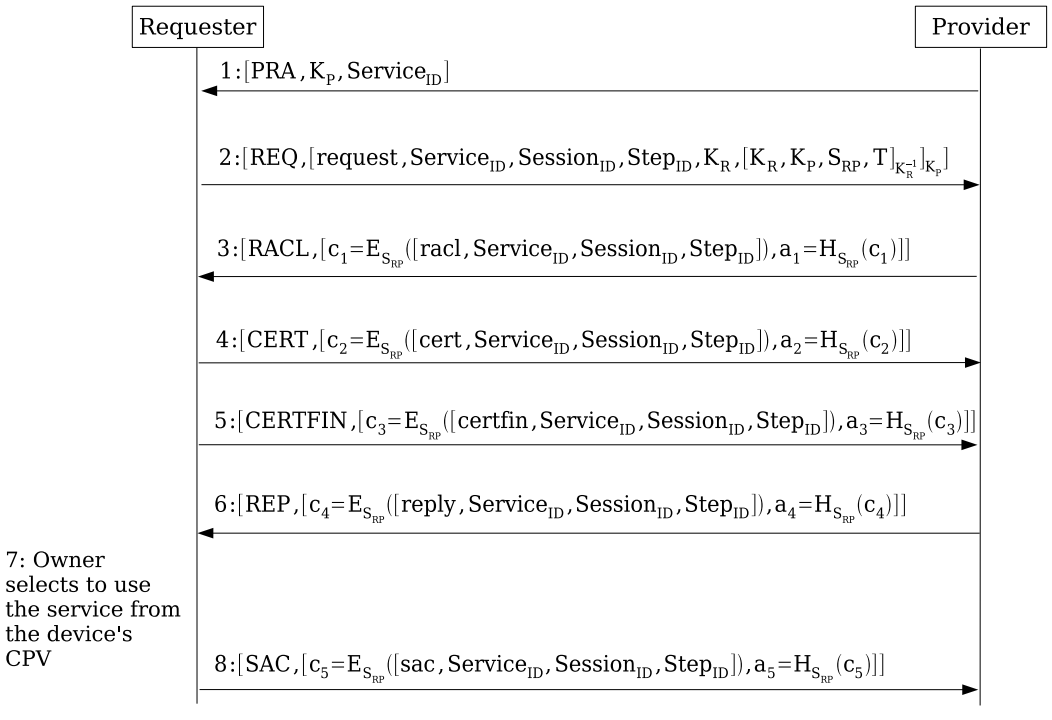

Figure 4.16: Eager service discovery and access protocol of ÆTHER$_1$.

REP message (like the one in step 6), in essence confirming the new session key. Now the requester can access the provided service with a *SAC* message (as in step 8) and after it does the session ends and the provider again deletes the symmetric key associated with this session (S'_{RP}). The entries of the authorization cache expire after a certain amount of time. This time period is directly related to the nature of the corresponding provided service. A typical value we suggest is ten minutes, a compromise between performance and the risk of granting access based on a previously valid AC that has expired during this time period. Another important issue has to do with context-sensitive permissions. When a permission requires the possession of context attributes, the service provider asks[11] the principal that is the source of authority for the specific context attribute[12] about the membership status of the requester. Authorization decisions that have been reached using context attributes are never cached. We have taken this design choice since the context of an entity may change at any time and therefore previously valid authorization decisions may no longer

[11] This is a normal access request performed according to the previously presented protocol.
[12] Identified through the CAS statements it locally has in its policy database.

be valid if the relevant context has changed.

4.4.5.2 Lazy

In the lazy access strategy pervasive devices remain passive regarding the service discovery process. Service providers broadcast their services and the pervasive device only responds when necessary. Hence, PRAs are not used. Instead service providers periodically broadcast DRAs for the services they want to offer. As we have described in the previous paragraph, DRAs contain a textual description of the corresponding service. Devices that receive DRAs enter them into their CPVs. When the owner of a device wants to access a particular service she lists the contents of the CPV maintained by the device she carries. Based on the descriptions she selects the appropriate service and the access protocol begins. This protocol is the same as the previous one with one difference. After the provider returns the authorization decision and the access interface of the service (assuming that the decision is positive) the requester directly sends a SAC message to use the service and the session ends. As in the eager strategy, we again rely on an authorization cache in order to improve performance. Fig. 4.17 presents the protocol.

The lazy strategy sacrifices the protection of service advertisements in order to avoid having devices in a PAD constantly trying to get access to all provided services. This helps devices to conserve battery energy, however it allows attackers to enumerate all services offered in a PAD. Whether a device operates in eager or lazy mode is an option that can be directly controlled by its owner.

4.4.6 Inference Engine

The inference engine of ÆTHER$_1$ takes as input the set of attribute credentials that a requester has sent to a provider in order to support an access request, the request itself and the set of local policies that a provider possesses. If there are authorizations that use context attributes the principals responsible for maintaining the corresponding dynamic sets are queried regarding the membership status of the requester. It provides as output a boolean value that represents whether the request is allowed to be performed or denied. Since the algorithm of the ÆTHER$_1$ inference engine has a lot of similar steps with the

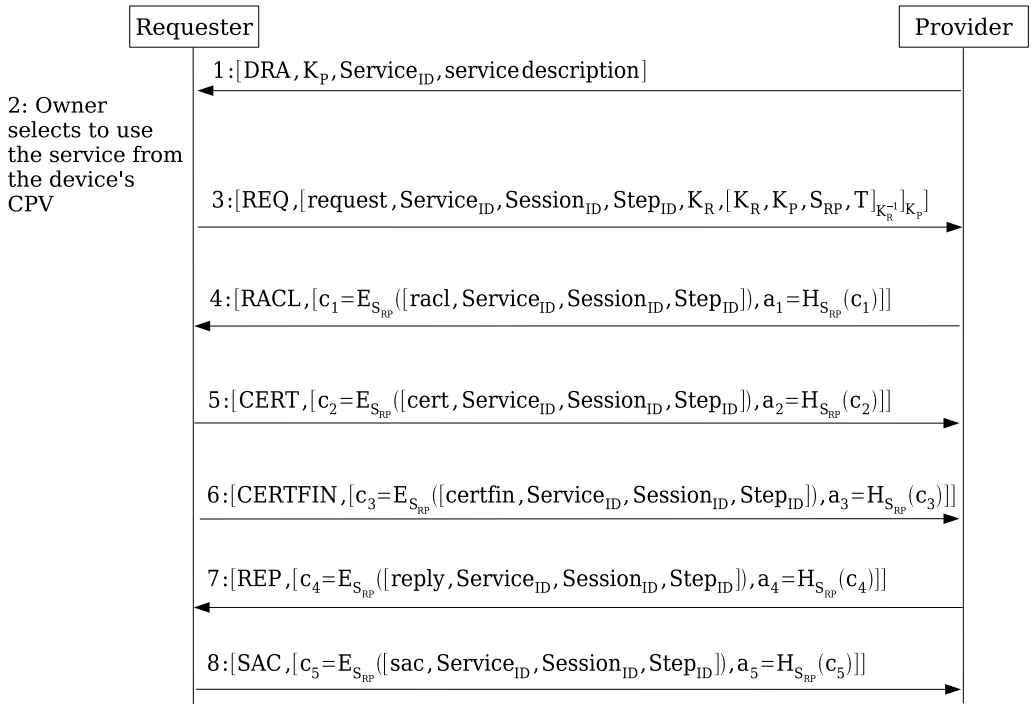

Figure 4.17: Lazy service discovery and access protocol of ÆTHER$_1$.

ÆTHER$_0$ inference engine algorithm we have previously described in paragraph 4.3.6, we only present here the steps that differ (however, Fig. 4.18 presents the complete decision flowchart of the ÆTHER$_1$ inference engine).

1. After a request is received the engine checks the authorization cache to find whether the requester has made the same access request previously. If it has it returns the decision from the cache.

2. The engine also checks the validity of the requester's ACs, i.e. whether they have been issued by members of the corresponding AASs or not. If they ACs are valid the decision value is set to *true* and returned. Otherwise it is set to *false* and returned.

4.4.7 Secure Interdomain Interactions

One of the goals of our architecture is to enable secure interactions between principals of different authority domains. While a user roams administrative domains the devices he carries must be able to interact with the devices of the environment in a secure manner

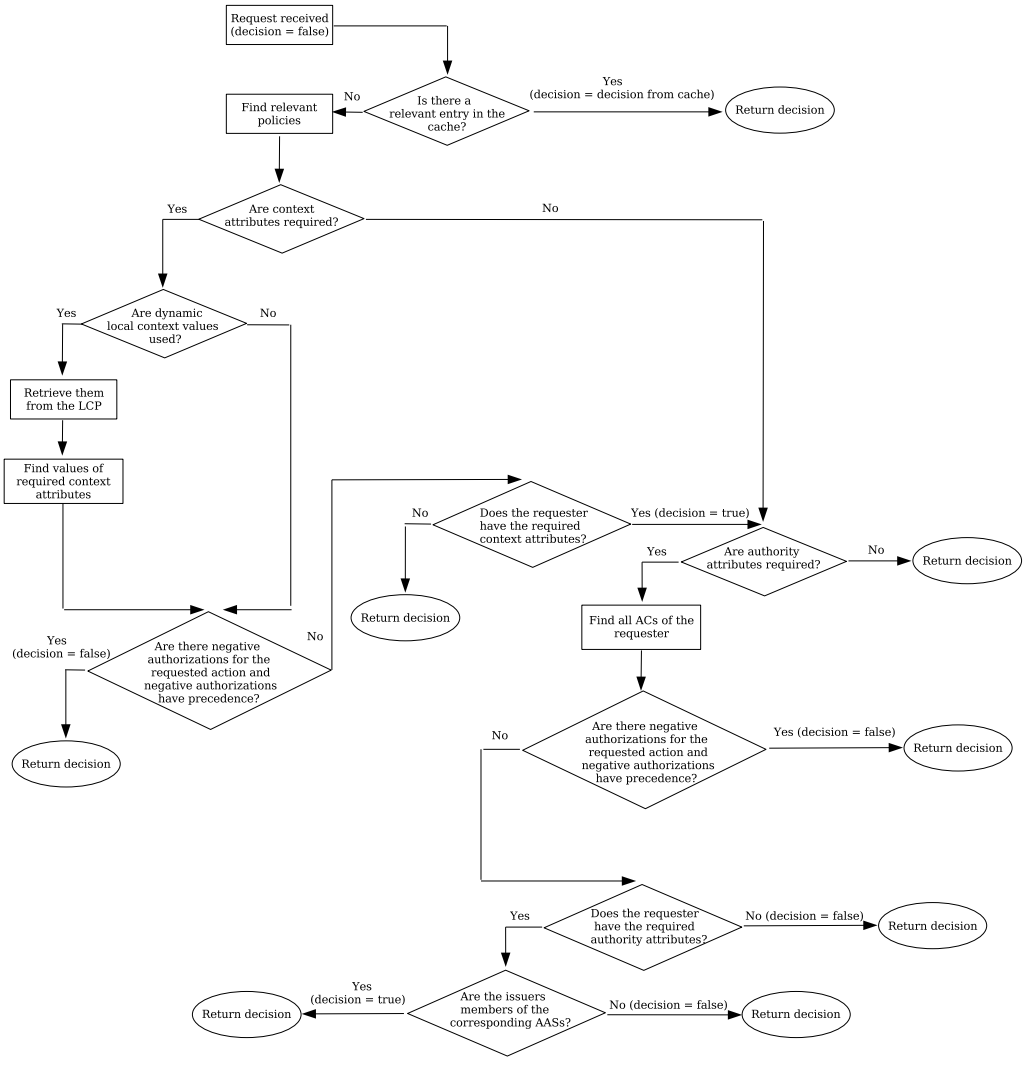

Figure 4.18: Access control decision flowchart of the ÆTHER$_1$ inference engine.

without requiring reconfiguration. When ACs are exchanged between different domains, attribute mapping mechanisms are needed to allow attributes from foreign domains to be translated into corresponding attributes in the domain where an AC is validated [LN99].

Although a global attribute registry has been suggested as a possible solution to the problem of attribute mapping [LN99], we believe that the introduction of a centralized universally trusted principal is not applicable to the highly distributed nature of ubiquitous computing. In ÆTHER$_1$ we use Attribute Mapping Certificates (AMCs) to map AASs of different PADs. As an example, consider the case of a business agreement between restaurants R_1 and R_2 that specifies that the privileged customers of R_1 are to be accepted as normal customers in R_2. This means that we want to map the authority attribute *(customer, privileged)* of R_1 to the authority attribute *(customer, normal)* of R_2. If the sources of authority for *(customer, privileged)* in R_1 are the principals A_1 and A_2 and for *(customer, normal)* in R_2 the principals B_1 and B_2 then the AMC that performs the mapping would have to be issued by B_1 or B_2. This AMC issued by principal B_1 is shown below:

aether version: 1
type: attribute mapping certificate
issuer: "3048024100d1d645e46841f0b708b75dd" (i.e. B_1's public key)
attribute name: customer
attribute value: normal
subject attribute name: customer
subject attribute value: privileged
subject sources of authority: "3048024100c", "3048024100a" (i.e. the public keys of A_1 and A_2)
not valid before: 2005/07/17-18:09
not valid after: 2005/07/18-17:09
signature: "4096816a21379e8d20d0206ff061f3af6a11" (i.e. B_1's signature)

Obviously this is a one-way mapping. If a two-way mapping is required then another AMC must be issued, by A_1 or A_2, to map the authority attribute *(customer, normal)* of R_2 to the authority attribute *(customer, privileged)* of R_1. Moreover, the mapping is between the sources of authority of the two AASs, meaning that ACs issued by principals that dynamically joined the mapped set are not accepted as valid in R_2.

4.4.8 Security Considerations

As in ÆTHER$_0$, the key agreement protocol we use in ÆTHER$_1$ as the basis for our access protocol relies on loosely synchronized clocks to protect against replay attacks. However, loosely synchronized clocks are required between the participants in any case since we use short expiration time periods for the issued ACs. Any other key agreement protocol that also provides secure transport services can be used instead of the one we have presented. For example, the SSL/TLS protocol can be used. In this case the requester and the provider generate self-signed X.509 certificates for their public keys. These certificates ascertain verifiers that the issuer indeed controls the private key that corresponds to the presented public key. After the end of the SSL/TLS key agreement and the establishment of the secure channel, the requester presents its ÆTHER$_1$ ACs to the provider normally. Before the provider passes these to its inference engine it makes sure that the *subject* field of the requester's ACs is the same as the public key contained in the requester's self-signed X.509 certificate used to establish the SSL/TLS channel. This is done in order to detect and avoid man-in-the-middle attacks. Afterwards, the authorization process proceeds as we have previously described. In fact, the application of ÆTHER$_1$ in the domain of web services uses this approach. For more details see paragraph 5.5.5.

Another security issue we have to address is that of checking whether a service provider is authorized to provide a particular service or not. For example, a requester may want to be sure that a file to be stored in a file server is going to reach an authorized server and not a rogue one. The ÆTHER$_1$ access control protocol only provides to the service provider the guarantee that the public key of the requester is allowed or not to access the service it offers. However, the requester is not able to know if the service provider is a legitimate or a malicious entity. In case the requester has to release information of sensitive nature while accessing a service (such as storing a personal file or printing an important document) it is important for the utilized protocol to be able to support *server-side* authorization.

This problem can addressed in ÆTHER$_1$ by extending the previously presented access protocols so that the requester demands the possession of certain authority or context attributes before it releases any kind of sensitive information to the service provider. The requester must construct authorizations for the resources it wants to protect. These au-

thorizations are regular ÆTHER$_1$ authorization statements. After the requester finishes sending its ACs to the provider and receives a positive reply for its access request, it sends its own RACL to the provider. Then the provider sends its own ACs to satisfy this RACL to the requester. The requester passes all these to its inference engine and if the output is positive then it sends the sensitive information to the provider who is now fully authorized.

4.5 Usability

One of the most difficult problems that every security solution for pervasive computing has to address is that of usability. Since pervasive computing aims to make the information processing capabilities of the environment integrate naturally with the workflow of the human users, it must not require from them to constantly answer complicated dialogs or fill input forms. Flaws in the human-computer interface design may result in the complete failure of otherwise perfectly secure systems. Users prefer to ignore, or even completely disable, security settings they do not fully understand [WT99]. For example, in early versions of ÆTHER we were querying the user whether a given previously established policy decision (like a binding or an authority attribute assignment) should be allowed to be renewed or not. However, we found out that this was particularly distracting and eventually we decided to fully integrate renewal mechanisms in our model.

Policy composition is another area of security usability that requires careful attention. We realize that only a very small percentage of users will actually fully explore all the functionalities of their devices and compose authorization policies for them. Users do not want to be bothered with managing their devices, yet they also do not want to leave their devices unprotected or let an outside, global and centralized entity to handle their management. To address this we suggest that manufacturers should provide default sets of policies with their devices according to the services they provide and the context in which they are going to be used. Users can then just create simple authorization policies (like "allow anyone in the room to use the printer", or "all my friends can access my mobile phone's files") probably with a graphical utility. Of course the ÆTHER policy language gives the option to power users to modify the default policies and create their own much more flexible and appropriate to specific situations policies.

Although the use of hexadecimal strings as identifiers for pervasive computing devices provides a level of privacy, it suffers from usability problems. It is especially difficult (if not impossible) for humans to memorize the identification strings of their devices. Hence, names are required in order to allow users to specify names instead of hexadecimal strings in policy specifications and interactions with their devices. The general ÆTHER model does not have an inherent understanding of principal names, ÆTHER_0 uses randomly (in part) generated values as identifiers while ÆTHER_1 uses public keys to identify participants. Hence, to support named principals, we need a naming mechanism.

In ÆTHER_0 alpha-pads can issue naming attribute assignments for the devices they bind. As an example consider the following attribute assignment issued by alpha-pad A to device B an MP3 player:

aether version: 0
type: attribute assignment
issuer: "3048024100cd462d1d7cf0aa50431616cb61ee" (ID_A)
subject: "3048024100d1d645e46841f0b708b75dd95851d" (ID_B)
attribute name: name
attribute value: my_mp3_player
not valid before: 2005/04/02-11:21
not valid after: 2005/04/02-12:21
renewable: 1

Now when alpha-pad A wants to issue a new policy or make an access request to device B the name can be used instead of the hexadecimal string. Alpha-pad A can make the conversion since it has the above statement that binds ID_B to the name *my_mp3_player*.

In ÆTHER_1 our AASs can be used to provide a mechanism to bind names to public keys. A PAD can have an AAS responsible for providing naming credentials, exactly like any other attribute. The only difference is that the value field of the attribute is not included in the AAS policy statement but is specified when the owner uses one of the member devices of the set to issue a naming credential. All the devices that are part of the domain and have the AAS policy statement that defines the naming attribute can validate naming certificates and perform the mapping between public keys and user-specified names. An example of an AAS policy statement that defines a naming attribute is shown below; a static AAS statement for a naming attribute, the keyword *<user-defined>* means that

the AC issuer, in this case this can only be principal Key_0, defines the attribute's value at the time of issuing,

aether version: 1
type: authority attribute set
issuer: "3048024100c44004f8f643f0b572ca85bb59f" (i.e. Key_0)
attribute name: name
attribute value: <user-defined>
membership threshold value: 0
delegation depth: 1
sources of authority: "3048024100c44004f8f643f0b572ca85bb59f" (Key_0)
signature: "dc7fb38cd89cb6422f98ce850718b96718" (Key_0's signature)

4.6 Maintainability

Maintainability refers to the ease with which any maintenance task can be carried out on an information processing device. Therefore, its role is essential in pervasive computing where human users have countless devices that have to be maintained in the most unobtrusive manner possible.

One of the maintainability problems that we have to consider is that of lost or stolen devices. In ÆTHER$_0$ when a user device is stolen it can either be an alpha-pad or a normal device. In the case it is a normal device then the problem can easily be addressed by the user. The secret keys the device has established with other devices during initiating or replying to service access requests will eventually expire and therefore do not present a direct system compromise. Of course if the stolen device is within the transmission range of other devices it has previously successfully communicated with then it can still be used to access their services until their shared keys expire. When the legitimate owner realizes that one of his ordinary devices has been lost or stolen, he can delete the relevant binding policy and the authority attribute assignments from the alpha-pad he used to bind it. However, if the stolen device is an alpha-pad then the problem becomes more complicated. The stolen alpha-pad was probably used by the user to bind several normal devices. The new owner (i.e. the thief) can now use the alpha-pad to have full access to every device it was used to create a binding with. If the legitimate owner was thoughtful enough to have bound his devices to more than one alpha-pads, then he can use one of these to revoke the

bindings of the stolen alpha-pad with a revoke binding operation. The other alternative is to use the hardware switch on the bound devices and completely delete all their policy state, completely losing any custom policies he has stored on them.

In ÆTHER$_1$ the problem of stolen devices is similar to the problem of stolen alpha-pads in ÆTHER$_0$. Since every device in ÆTHER$_1$ is allowed to bind other devices, a stolen device means that the owner must revoke all the bindings that the stolen device has been used to create. Again, this can be accomplished either with the hardware switch that wipes all policy state from a bound device, or with another device that has been used to create an additional binding with the target device. We do not believe that multiple bindings are an inconvenience to end users. When a user has a device that is of particular importance and has a significant chance to be stolen (due to its size and the fact that the user always carries it with him) it makes sense not to bind all other devices in a PAD just with this particular device.

Another maintenance problem that must be addressed is that of lost configuration information from devices due to discharged batteries. The importance of this problem became apparent to us during the development of ÆTHER. The handheld devices we were using for implementation and testing lacked non-volatile memory and stored all configuration information in RAM. When their batteries were not charged for extended periods of time the devices lost all configuration state returning to their default factory settings and we had to reconfigure them. Hence, a recovery process must exist in order to allow users to save the custom ÆTHER policies they have created for their devices and reinstall them without extensive configuration tasks. We accomplish this by periodically (for example once every week or more frequently depending on the importance of the device) sending all policy data a device has to the device that bound it. This applies to both instantiations of our model. If a device was bound by more than one devices then its policy data are sent to all such devices. This allows the replication of the backed up policies. These transfers are trivial to secure in both ÆTHER$_0$ where the master and the slave device share a secret key, and in ÆTHER$_1$ where the devices have exchanged public keys during their binding procedure.

4.7 Privacy Considerations

Although the protection of privacy is not one of the primary goals of our architecture, we have to consider possible threats to the privacy of the participating entities. Privacy concerns are going to play a central role in the adoption of pervasive computing systems and technologies [Lan01].

The direct use of hexadecimal identifiers (randomly generated in ÆTHER$_0$ and encoded public keys in ÆTHER$_1$) in the attribute assignments, the credentials and the access requests of our architecture provides a level of privacy since the identity of the related principals is not revealed. However, the structure of trust relationships in a PAD can be disclosed by a malicious entity that observes these exchanges. Since a service provider receives the requester's identifier, the requester can still be tracked via the use of this identifier. In order to address this we propose the use of *profiles*. A user can have many profiles on a device, each one with its own identifier and its own set of policies. The user can force the device she is about to use to generate a new identifier each time she visits a new pervasive computing environment. In ÆTHER$_0$ this is simply a randomly generated number, while in ÆTHER$_1$ a new public key pair. Therefore, in ÆTHER$_1$ each time a user visits a new environment she can instruct the device she is about to use to generate a new public key pair and use this one for getting certificates and accessing services. Since ACs expire after an hour, it makes no sense from a privacy perspective for the user to keep a repository with expired certificates on her device. In ÆTHER$_0$ alpha-pads are the central devices that manage the security relationships that a user establishes. An alpha-pad profile includes not only the identifier of the alpha-pad itself and its related policies, but also all the identifiers and the policies of all the devices it was used to bind. Consequently, changing the profile of an alpha-pad results in all devices bound by this alpha-pad to change to the new profile. The command to change to the new profile can be issued from an alpha-pad to all the devices it has bound via a wireless transmission secured with the respective secret shared key.

4.8 Comparison

The main difference between the ÆTHER$_0$ and the ÆTHER$_1$ instantiations is that they follow different management models. While ÆTHER$_0$ does not depend on any external, globally trusted, always available and centralized servers it adopts a locally centralized model. All security relationships must be established through the user's alpha-pad (or alpha-pads if the user has more than one) which must always be carried around. This locally centralized model is easy for users to understand and manage; a user is fully aware that in order for another person to unlock the door of his office that person must be authorized with the user's PDA which plays the role of his alpha-pad. Since all authority flows to and from the alpha-pad, users can easily understand and therefore manage without difficulties ÆTHER$_0$ PADs and their interactions with the PADs of other users. Furthermore, the locally centralized model allows us to use symmetric cryptography which requires significantly less computational resources than asymmetric cryptography. ÆTHER$_0$ is appropriate for small PADs with no more than ten to twenty participating devices; more than that the use of symmetric cryptography becomes cumbersome. The main disadvantage of ÆTHER$_0$ is that the alpha-pad represents a single point of attack and failure. Since all security data are stored in alpha-pads they become particularly attractive targets to potential digital attackers and physical world thieves. The main argument against this is that users are used to carry around valuable physical objects, like mobile phones and credit cards, and therefore know how to protect them from being stolen. Regarding digital attacks, alpha-pads must be engineered as high-value software entities in order to be able to withstand them.

ÆTHER$_1$ follows both a globally and a locally decentralized management model. All participating devices have their own namespace and are free to bind other devices, define policies for them, and issue attribute assignments in the form of attribute certificates. Moreover, the management model of ÆTHER$_1$ allows the sets of principals that act as sources of authority for specific authority attributes to grow dynamically. This approach overcomes the traditionally centralized nature of RBAC and enables the establishment of trust with unknown principals authorizing their actions in the local domain. The ÆTHER$_1$ instantiation relies on asymmetric cryptography and therefore is appropriate for devices

that are able to support its computationally expensive algorithms. However, the constant advancements in microprocessor design suggest that pervasive computing devices of the (not too distant) future will not have problems with such computational or memory requirements. The main advantage of ÆTHER$_1$ over ÆTHER$_0$ is that it does not rely on any locally (or globally) centralized entity. A user can simply pick up any of his device and issue certificates to other users he encounters, bind new devices, or use it to access provided services. There is no single point of failure in ÆTHER$_1$ PADs, however this makes their visualization by users difficult. Since authority can flow from and to many different devices, users with small PADs and simple requirements may find this fully decentralized model difficult to understand and manage. Hence, ÆTHER$_1$ targets large environments which consist of many devices whose ownership rights may not be clearly defined. Workplaces shared by many people, such as business offices for example, and commercial avenues, such as restaurants, are the natural application environments of ÆTHER$_1$. Table 4.1 summarizes this discussion.

Table 4.1: Comparison of ÆTHER$_0$ and ÆTHER$_1$.

Characteristics	ÆTHER$_0$	ÆTHER$_1$
Management model	Globally decentralized, locally centralized.	Globally decentralized, locally decentralized.
Disconnected operation	Globally yes, locally no.	Globally yes, locally yes.
Namespace	Each alpha-pad has its own namespace.	Each device has its own namespace.
Cryptography	Symmetric.	Both; asymmetric for ACs and key agreement, symmetric for bulk data transfer.
Authority attribute assignments	By alpha-pads, assignments stay local to the issuer.	By any device that is a member of the corresponding AAS, assignments are ACs given to subjects.
Revocation	Short validity periods.	Short validity periods.
Context-awareness	Supported.	Supported.
Secure service discovery	Supported.	Supported in eager strategy, not supported in lazy strategy.

Chapter 5

ÆTHER Implementation

In the previous chapter we have presented the design of the general ÆTHER architecture and the details of instantiating it into two different management models, namely ÆTHER_0 and ÆTHER_1. This chapter presents the implementation of two prototypes for these instantiations and examines their feasibility using modern handheld devices as the development hardware platform. Both of our prototypes are implemented as *layers* of the Networks and Telecommunications Research Group (NTRG) ad hoc networking stack developed at the University of Dublin, Trinity College [OD01]. In the NTRG stack components are assembled using a layered architecture achieving abstraction borders between the different implementation elements. The reasons we selected it for implementing ÆTHER are:

- The NTRG stack includes layers that implement several building blocks of mobile ad hoc networks, such as routing protocols, autoconfiguration addressing schemes, and decentralized location services, that can be directly reused for setting up the communications infrastructure of pervasive computing environments. This fact allows us to focus solely on the implementation of ÆTHER.
- The NTRG stack is simple and extensible allowing us to quickly build and evaluate the service discovery and access protocols of ÆTHER.
- It has been designed to work on modern handheld devices giving us the opportunity to experiment with real-world pervasive computing scenarios and applications.

The first section of the chapter gives a brief explanation of the NTRG stack, its layers and the related issues. The second section presents a detailed examination of the ÆTHER layer and how are the two instantiations of the general model implemented. These are followed by a section in which we discuss an experimental evaluation of ÆTHER_0 and ÆTHER_1. The chapter concludes with describing applications of the two instantiations, both practical ones that we have fully developed and futuristic ones that we only discuss theoretically to assess our models.

5.1 The NTRG Ad hoc Networking Stack

The concept of a protocol stack is well-defined in the computer networking literature; instead of following a monolithic approach to the process of designing and developing network communications, the required functionalities are divided into distinct *layers* that implement a specific component of the whole process. An ordered collection of such layers is called a *stack*. Although layers pass information directly to the layers above or below them in the stack, each layer communicates in a logical *end-to-end* manner with each layer of the same type on stacks of different networked entities. The most well-known networking stack is the ISO reference model for Open Systems Interconnection (OSI) [ISO84].

The NTRG stack follows the same approach but focuses on ad hoc networking protocols. At the physical layer it provides implementations of the IEEE 802.11b Wireless Local Area Network (WLAN) protocol, the Bluetooth protocol and infrared. At the routing layer the Dynamic Source Routing (DSR) [JMB01] and the Ad hoc On-demand Distance Vector (AODV) [PR99] protocols have been implemented as well as other less commonly used ad hoc routing protocols. Furthermore, the NTRG stack includes a dynamic addressing layer [TO03] and several application layers like instant messaging, voice transfer (phone), and a web gateway layer among others. The main goal of the NTRG stack is "to produce a general-purpose mobile node capable of running a large range of network applications that can adapt its mode of operation to the prevailing wireless network architecture" [OD01]. As a simple example consider the stack presented in Fig. 5.1 a). The stack has been implemented using the C++ programming language for Windows-based operating systems. It currently supports Windows 98, Windows 2000, Windows XP and Windows CE. This

gives us the opportunity to test our implementation in a wide variety of hardware platforms; from ordinary desktops to laptops, handhelds and smart phones among others. We must also note that most of the stack code is highly portable. Hence, it can easily be modified to support other platforms given that threading, file manipulation and networking libraries are provided by the hosting operating system.

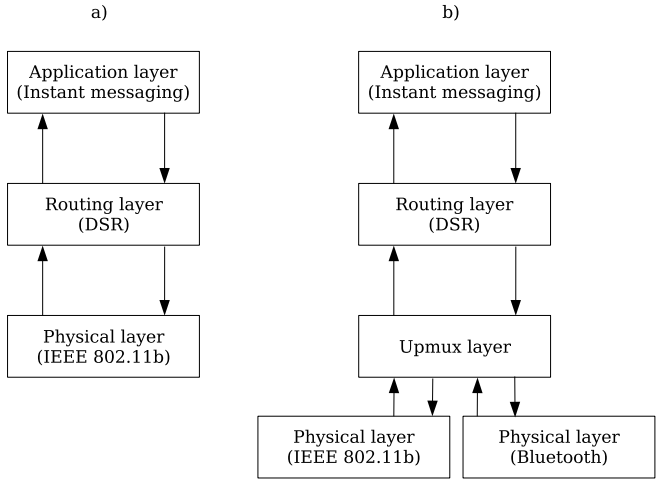

Figure 5.1: The NTRG ad hoc networking stack.

Another facility provided by the stack is the *upmux* layer; a layer to do upward multiplexing. It allows multiple independent layers to sit below a single stack layer (see Fig. 5.1 b)). This finds a lot of applications, for example when we want to create network entities that bridge two distinct domains built on different wireless networking technologies. Fig. 5.2 presents the ÆTHER layer as part of the NTRG stack. In the following paragraphs we will discuss the utilized layers and the considerations associated with the implementation of ÆTHER.

5.1.1 The Physical Layer

At the physical layer level of the stack we depend on two different layers; the IEEE 802.11b WLAN layer which we use as the ordinary wireless communication interface and the infrared layer which we use as the location-limited channel interface. In order to implement the required dual interface we utilize the upmux layer for doing upward multiplexing. ÆTHER packets that come from above and have to be transmitted over a location-limited

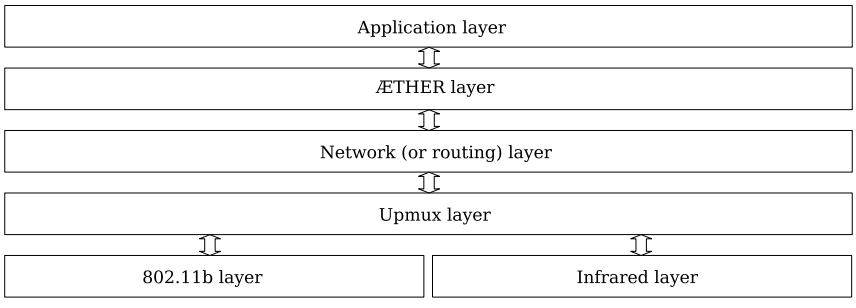

Figure 5.2: The ÆTHER layer as part of the NTRG stack.

channel are labelled in order to bypass the routing layer and be directly forwarded to the infrared layer. Please note that the bypassing of the routing layer is done purely for semantic reasons; even if routing was utilized it would not make any difference since location-limited channels are strictly one hop, i.e. always between two entities of which one is the sender and the other the receiver. The operation of the infrared layer is simple. It waits for data from infrared peers and forwards them upwards the stack. When data arrive from above layers, the infrared layer tries to discover infrared peers within range and sends to them these data. If none are discovered the data are discarded and an error notification message is sent upwards.

5.1.2 The Network Layer

At the network layer any routing protocol can be utilized. We believe that ad hoc routing protocols, like DSR and AODV for example, are fully compatible with the decentralized and mobile nature of pervasive computing. Although we advocate their use, the operation of ÆTHER does not depend on any specific routing protocol. In our current implementation we use DSR as it is one of the most mature and widely deployed solutions. For routing identifiers (i.e. node IDs) we use quantities derived from the entity's ÆTHER identifier. In ÆTHER_0 entity identifiers are directly used and in ÆTHER_1 we use the digital hash of the entity's public key. These routing identifiers have the following two properties [CM02], [OR01]:

- They are statistically unique since they were generated with a cryptographic hash function that is collision-resistant. In ÆTHER_0 the input to this function is the

concatenation of the hardware addresses of the device's two interfaces (WLAN and infrared) and a randomly generated number. In ÆTHER$_1$ the entity's public key is used as input.

- They are securely bound to a given entity. This is accomplished differently in our two instantiations. In ÆTHER$_0$ when an entity learns another entity's identifier its authenticity is guaranteed since this process takes place over a location-limited channel. After this initial establishment the two entities share a symmetric secret key and therefore they can easily authenticate each other. In ÆTHER$_1$ an entity can prove the ownership of the routing identifier it uses since this is derived from its public key and can therefore simply sign messages with the corresponding private key which it controls.

One possible attack against this scheme can be accomplished by utilizing recent advances in hash collision search techniques. At the 24th Annual International Cryptology Conference a number of hash collision attacks were announced against the SHA, MD4 and MD5 algorithms (among others) [BC04], [WFLY04]. Furthermore, it has recently been shown that SHA-1 is also not collision-free [WYY05]. There are two possible attack avenues against our use of hashes as routing identifiers. In ordinary collision attacks the attacker is able to find two messages that produce the same hash, but has no control over what the hash can be. Therefore, this has limited impact on our approach. On the other hand, pre-image attacks allow the attacker to find an input message that causes a hash function to produce a particular output. In this case the attacker is able to forge a routing identifier and impersonate the rightful owner, causing mainly denial of service attacks. The solution is to use a hash function that does not suffer from such problems, like SHA-256.

Another aspect we have to consider at the network layer is the use of address autoconfiguration management schemes, like for example the one that has been implemented for the NTRG stack and presented in [TO03]. Such mechanisms are used to manage the addressing requirements in cases where we have the merging of domains to form a larger domain, or when a domain is split to a number of smaller domains, possibly due to network partitioning. In these situations the autoconfiguration mechanism needs to change the network identifiers of the entities in order to guarantee that they are still connected

to the network and participating in the routing process. However, these identifiers are not randomly chosen in ÆTHER but are directly related to each entity (as we have described above) and therefore cannot be arbitrarily changed. We address this problem by semantically dividing the utilized identifiers into two parts. For example, when 160-bit identifiers are used the management of the leftmost 80 bits is given to the address autoconfiguration scheme that keeps this part of the identifier routable at all times. The rightmost 80 bits are the ÆTHER identifier and remain constant. Our approach is equivalent to the one used in IPv6 for similar purposes [OR01].

5.1.3 The Application Layer

This layer implements the application-level services that are offered in a pervasive computing environment. The NTRG stack has a lot of example applications, and a number of them have been modified to be secured using ÆTHER. For specific examples please see section 5.5 in this chapter. Since service discovery is handled as part of our architecture, the first step in making an application ÆTHER-enabled is to compose authorization policies for the services that need to be provided. Based on these policies the ÆTHER layer advertises the services[1] and plays the role of a reference monitor for each access request that arrives from below. After a request has been authorized the application layer of the service provider is forwarded the request by the ÆTHER layer below it. Every message that is sent down the stack after this point is secured using the key material that have been negotiated by the ÆTHER layer during the authorization process. The ÆTHER layer of the requester receives these messages, decrypts and authenticates them using the negotiated key, and forwards them to the application layer above it.

5.2 The ÆTHER Layer

ÆTHER_0 and ÆTHER_1 have been implemented as two distinct layers for the NTRG stack. However, since both of them share the same generic model a lot of software components have been reused in their implementation. Fig. 5.3 illustrates the different components of

[1] Service advertisement works differently in our two instantiations as we have already presented in the design chapter.

the ÆTHER layer.

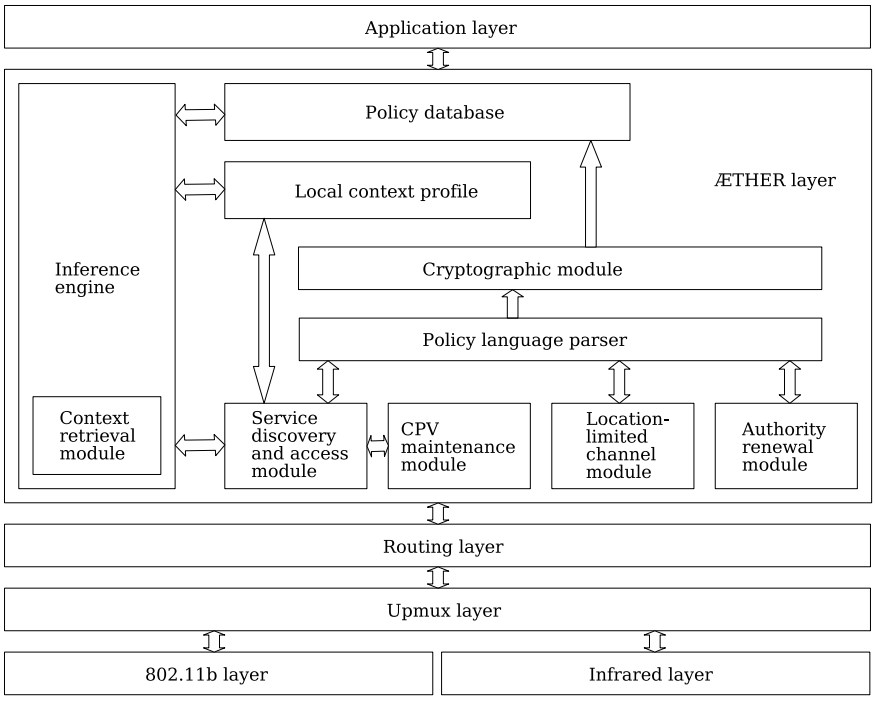

Figure 5.3: The ÆTHER layer and its components.

The ÆTHER layer is situated between the application and the network layers acting as an authorization middleware component and handles the process of transmitting and receiving service access requests and replies. In the following paragraphs we will analyze its components and describe the related implementation details of the two instantiations.

- Location-limited channel module. This module of the ÆTHER layer is responsible for receiving from below and sending downwards location-limited channel messages. These are labelled in order to bypass the routing layer. Moreover, the label is identified by the upmux layer that forwards them to the infrared layer.

- Authority renewal module. This module handles the incoming requests for authority renewals. In the ÆTHER_0 layer these are requests for the renewal of previously established mutual bindings or attribute assignments that have been sent to devices bound by the current alpha-pad device. In the ÆTHER_1 layer incoming renewal requests concern ACs previously issued by the current device to a subject device

over a location-limited channel.

- The CPV maintenance module keeps updated the list of services that have been discovered in the local environment. The update strategy follows different algorithms in the implementations of the ÆTHER$_0$ and ÆTHER$_1$ layers. This module communicates directly with the service discovery and access module from which it receives information regarding the discovered and accessed services. The data kept in the CPV are used by the implemented interaction interface that presents to the user of the device the services and allows her to select which ones she wants to use.

- The service discovery and access module implements the protocols we have presented in the design chapter responsible for receiving access requests and replies. Furthermore, this module handles session establishment and maintenance, as well as the management of key material used for protecting the transmitted messages. Reliability is implemented by using a retransmission mechanism that maintains a timer and keeps retransmitting a message until the expected reply is received. To detect replayed datagrams we use timestamps as we have explained in the design chapter. In the ÆTHER$_1$ layer this component also maintains a cache of recently exchanged authorization information in order to optimize the trust establishment procedure. Cache maintenance depends on the decisions of the inference engine and on session management.

- The policy language parser is implemented using Lex for lexical processing and Yacc for defining the grammar of the language [LMB92]. The ÆTHER policy language is heavily based on the KeyNote assertion grammar, however it introduces the required constructs to implement the ÆTHER-specific policy data structures. This module is instantiation-independent since it includes support for all the policy statements used by both ÆTHER$_0$ and ÆTHER$_1$.

- The cryptographic module is also the same in both layers that implement the two instantiations of our general model. Its responsibility is to provide a high-level Application Programming Interface (API) for the required cryptographic operations used by ÆTHER. It has been implemented using the OpenSSL [SSL05] open source

cryptographic toolkit. ÆTHER uses the AES (Rijndael) algorithm for all symmetric cryptographic operations with 256-bit keys. For asymmetric operations we rely on the RSA algorithm and we use key sizes of 512 and 1,024 bits. For message digesting we are currently using SHA-1. However, this was shown to be vulnerable to collision attacks and therefore should be replaced with SHA-256[2] or any other sufficiently strong message digesting algorithm. The modular design of the cryptographic module allows us to easily perform such replacements of the underlying utilized algorithms.

- The Local Context Profile (LCP) component maintains a list of the context attributes that apply to the current device. The values of these attributes are retrieved from the LCP when a value of a context attribute used in a policy statement has been defined as a dynamic local context value (i.e. has a preceding underscore). The LCP is implemented as a two-way linked list that contains elements of the structure that is shown below. Each context attribute of the LCP is associated with a Context Retrieval Function (CRF). The CRF implements the way that the value of a specific attribute is retrieved when requested from the LCP. For example, if we have a context attribute *(current_ time, _value)* used in a policy statement its CRF in the LCP would simply use the hosting operating system's facilities for getting the current time. For this attribute the *source* variable of the structure would have the NULL value. On the other hand, if the attribute is maintained by another entity, like for example a location sensor, the *source* variable would contain the identifier of the maintainer. The related CRF would send an access request to this principal in order to get a value for the attribute. The result of the CRF is saved in the *value* variable. We have implemented two example CRFs; one that simply retrieves the current time from the local device using the *localtime()* function, and one that makes an access request to a remote principal regarding the value of an attribute.

```
typedef struct context_attribute ctx_attr;
struct context_attribute
{
 char *name; /* name of the context attribute */
 char *value; /* its current value */
```

[2]Which is significantly slower.

```
  char *source; /* source(s) of authority, comma separated */
  void (*crf)(void *args); /* context retrieval function, or CRF */
  void *arg; /* the first argument of the CRF */
  ctx_attr *next; /* next element in the list */
  ctx_attr *prev; /* previous element in the list */
};
```

- The policy database contains all the ÆTHER statements trusted by the current device. For example, these include policies issued by the entity that bound the current device, or in the ÆTHER_1 layer ACs sent by another entity to support a service request that have been verified by the cryptographic module. The policy database is also implemented as a two-way linked list.

- The inference engine module is different in the ÆTHER_0 and the ÆTHER_1 layers. Each implementation realizes the decision flowchart we have presented in the relevant sections of the design chapter. A subcomponent of the inference engine of both implementations is the context retrieval module. The responsibility of this module is to find the CAS policy statement for a specific context attribute used in a policy statement and query the entity that maintains the relevant dynamic set about the membership status of the requester. This is an ordinary access request that is handled by the service discovery and access module.

Fig. 5.4 presents a high-level example of a scenario involving two ÆTHER_1 layers. Device A is a service provider; also it issues the necessary ACs to device B over the infrared channel. Device B then makes an access request over the 802.11b channel and receives a reply.

5.3 Performance Analysis

In order to evaluate the two instantiations of our architecture in a quantitative manner we have completed a detailed performance analysis based on modern handheld devices. Specifically, the hardware platform we use is the HP iPAQ H6340 [IPA05] with a Texas Instruments OMAP 1510 processor at 168 MHz and 64 MB RAM (64 MB ROM), running the Windows CE Mobile 2003 [WCE05] operating system. As we have already mentioned,

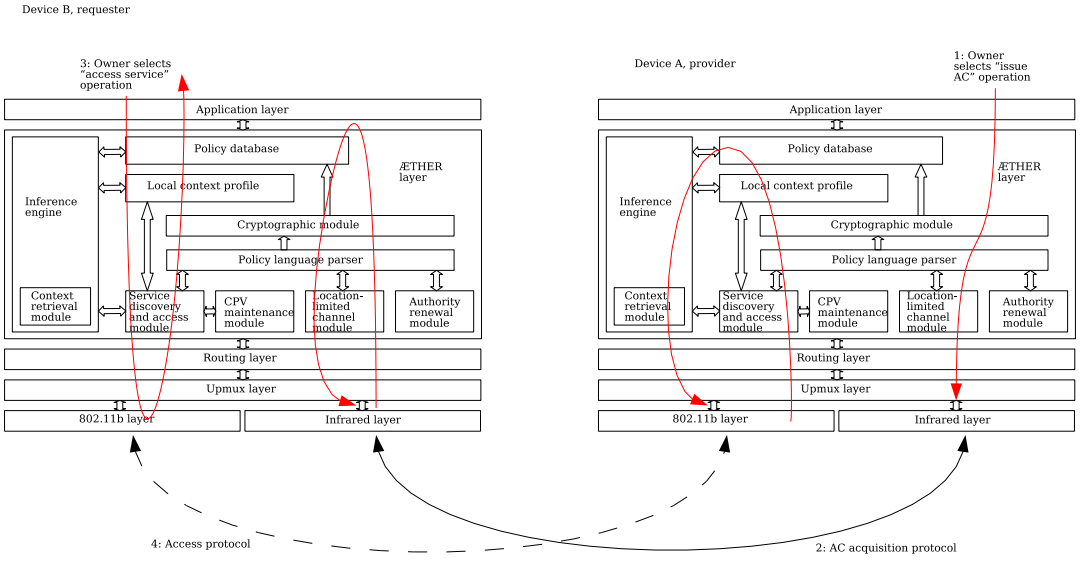

Figure 5.4: High-level example involving two ÆTHER$_1$ layers.

we use the infrared interface of the handhelds to implement the required location-limited channel and the IEEE 802.11b WLAN interface for normal communication links. We must also note that no compiler optimizations were used during the compilation of the test programs, and the thread priority during their execution was normal.

5.3.1 ÆTHER$_0$

In the ÆTHER$_0$ instantiation we have evaluated the process of mutually binding two alpha-pad devices over an established infrared channel, the following access protocols: normal device to alpha-pad, normal device to normal device (same alpha-pad), and normal device to normal device (different alpha-pad) as presented in paragraphs 4.3.5.1, 4.3.5.2, and 4.3.5.5 respectively, and the inference engine overhead introduced by the utilized number of attributes in authorization policies.

5.3.1.1 Mutual Binding

The mutual binding protocol in ÆTHER$_0$ takes place between two alpha-pad devices according to the protocol we have described in paragraph 4.3.3. The owner of alpha-pad B selects the "initiate mutual binding" operation on his device and the owner of alpha-pad

A accepts it. We have measured the time required for the protocol to complete from the perspective of alpha-pad A; from when it receives the initial $MBREQ$ message until the point it receives the $MBACK$ message and checks that the established symmetric key was not corrupted during its transfer over the infrared communication channel. An example of an established mutual binding policy statement between the two alpha-pads is the following (of course the *shared secret* field is different at each iteration since it is randomly generated by A):

aether version: 0
type: mutual binding
issuer: "3048024100cd462d1d7cf0aa50431616cb61" (The complete field is 40 characters long)
subject: "d0d331972525554c54f1e32b85bbf827240" (The complete field is 40 characters long)
not valid before: 2005/09/02-12:21
not valid after: 2005/09/03-11:21
shared secret: "ab56a13a94ebdef6c8e083d721df8e" (The complete field is 64 characters long)

The average time required for mutually binding two alpha-pads is 0.592 seconds (592 milliseconds), without measuring the time required for the two users to initiate and accept the process through their devices' interaction interfaces (see Fig. 5.5).

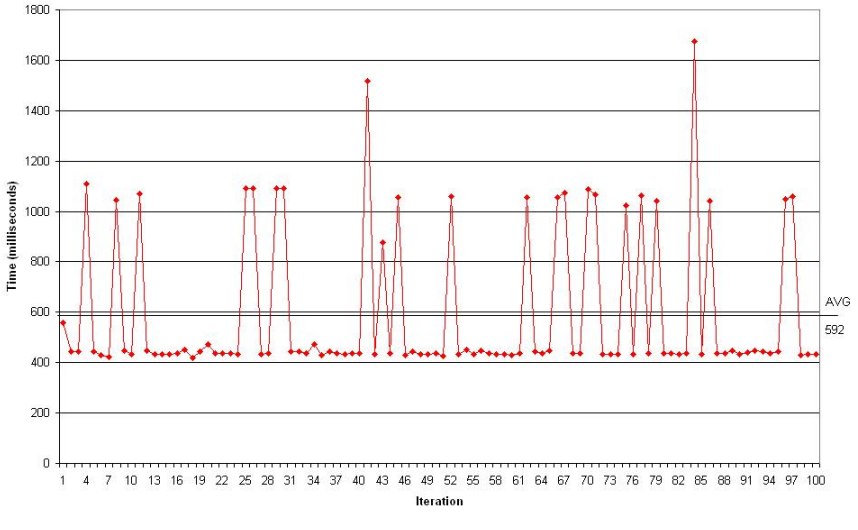

Figure 5.5: ÆTHER$_0$ mutual binding timing measurement.

We believe that at just over half a second the infrared-based mutual binding protocol of ÆTHER$_0$ can easily be integrated with the actions of the human owners of the participating

devices without introducing significant distractions.

5.3.1.2 Access Protocols

We have also evaluated ÆTHER$_0$ in respect to the time required for the completion of the different access protocols we have presented in paragraph 4.3.5. In all the experiments we use AES with a 256-bit key (in CBC mode) as the symmetric encryption algorithm and HMAC-SHA-1 as the HMAC algorithm. Furthermore, we assume that the authorization policy for the requested operation specifies only one required authority attribute. The one we use for all the experiments is the following:

```
aether version: 0
type: positive authorization
issuer: "3048024100ab3ddca94cef74c5fe2e4e7330bb5a1"
resource: switch
operation: change_state
requires: (@group == family_member);
```

We also assume that the alpha-pad (or the alpha-pad that bound the device that offers the requested service in the last two experiments) has assigned to the owner of the requesting device the required authority attribute and therefore reaches a positive decision. In the normal device to alpha-pad protocol (presented in paragraph 4.3.5.1) we have measured the time required for a requester (the normal device) to construct a request (message *REQ*), send it over the wireless interface to the alpha-pad that provides a service, and receive a reply (message *REP*). The average time required for the protocol to complete is 0.175 seconds (175 milliseconds). The second case examines the normal device to normal device (same alpha-pad) access protocol that we have analyzed in paragraph 4.3.5.2. Again we have measured the time required for the requesting device to send a *REQ* message to the alpha-pad that bound it, for the alpha-pad to forward the request and the authority attribute assignments it has for the requester to the target device, and the target device to reach a decision and transmit it to the requester with a *REP* message. The required average time for the completion of this protocol is 0.189 seconds (189 milliseconds). In the final access protocol experiment for ÆTHER$_0$ we have examined the situation where a request is initiated from a normal device to another normal device. The assumption is that

the alpha-pads that bound the devices share a mutual binding. This protocol has been presented in paragraph 4.3.5.5. The average time for a requester to send a *REQ* message and receive a reply in a *REP* message is 0.247 seconds (247 milliseconds) in this case. All three access protocol measurements for ÆTHER$_0$ are presented in Fig. 5.6.

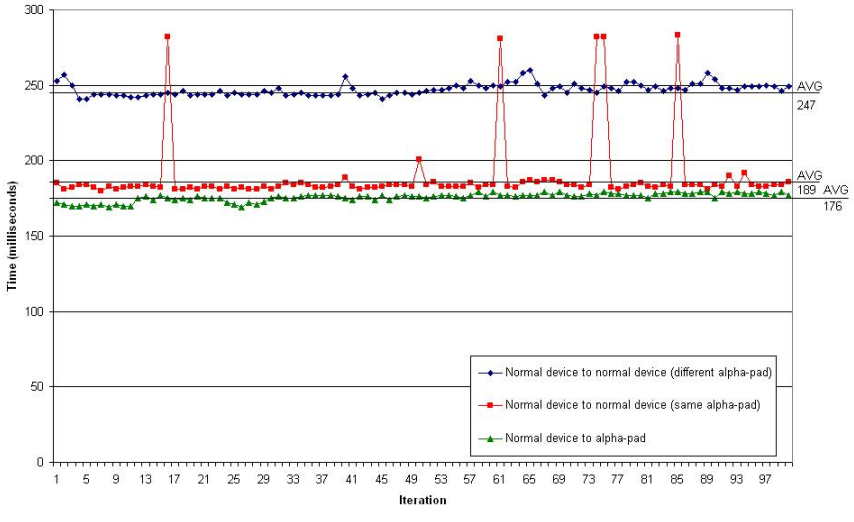

Figure 5.6: Timing measurements for the ÆTHER$_0$ access protocols.

The timing measurements of the ÆTHER$_0$ access protocols demonstrate the feasibility of using them on mobile constrained devices. Even the normal device to normal device (different alpha-pad) protocol which is the most complicated of the three and involves message exchanges between four different devices requires approximately only 0.25 seconds in average. Based on previous work done on benchmarking cryptographic algorithms [LDH04], like the AES cipher we use, we believe that the ÆTHER$_0$ instantiation can easily be deployed even in low-end sensor networks consisting of devices with 8-bit processors.

5.3.1.3 Policy Evaluation

In order to evaluate the inference engine of the ÆTHER$_0$ instantiation we have performed successive experiments with an increasing number of required authority attributes in the authorization policy that applies to the requested action. In this experiment we have only measured the time required for a service provider to reach an authorization decision, without taking into consideration message exchanges over the wireless communication medium.

The results are shown in Fig. 5.7 and are the averages of one hundred iterations for each experiment.

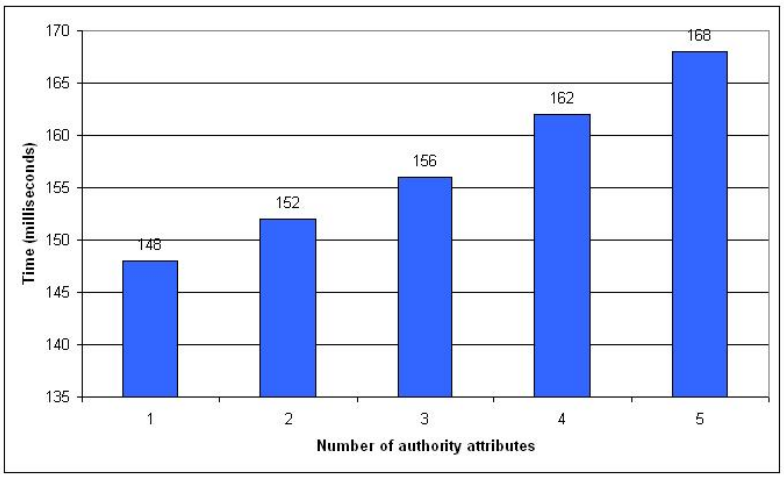

Figure 5.7: Timing measurements for the ÆTHER$_0$ inference engine.

When only one authority attribute is required the average time for the inference engine to reach an authorization decision is 0.148 seconds (148 milliseconds). As the number of required authority attributes increases, so does the average time needed for the inference engine to reach a decision, by an average of 5 milliseconds. For five authority attributes the required time is 0.168 seconds (168 milliseconds). Therefore, the overhead introduced by the ÆTHER$_0$ inference engine is not prohibitive for pervasive computing applications.

5.3.2 ÆTHER$_1$

Our evaluation of the ÆTHER$_1$ instantiation focused on the following processes: the binding of one device by another over the infrared channel, the authority attribute acquisition protocol that also takes place over the infrared channel, the wireless-based access protocol described in paragraph 4.4.5.1, and the overhead introduced by the inference engine. All the experiments were performed with RSA keys of 512 and 1,024 bits size, with small public exponents (e was given the value 65,537) making the public key operations significantly faster than the private key operations.

5.3.2.1 Binding

The binding procedure of ÆTHER$_1$ requires the establishment of an infrared channel between two devices; the protocol has been described in detail in paragraph 4.4.3. On the device to be bound, B, the owner selects the "bind" operation. Device B generates and sends a $BREQ$ message to device A with which an infrared channel has been established with the help of the human owner of the devices. Device A generates a binding policy and sends it to B which verifies A's signature. In this test we have measured the time required for such a binding from the perspective of the bound device, that is B. Fig. 5.8 shows the timing measurements with both 512 and 1,024 bits key sizes.

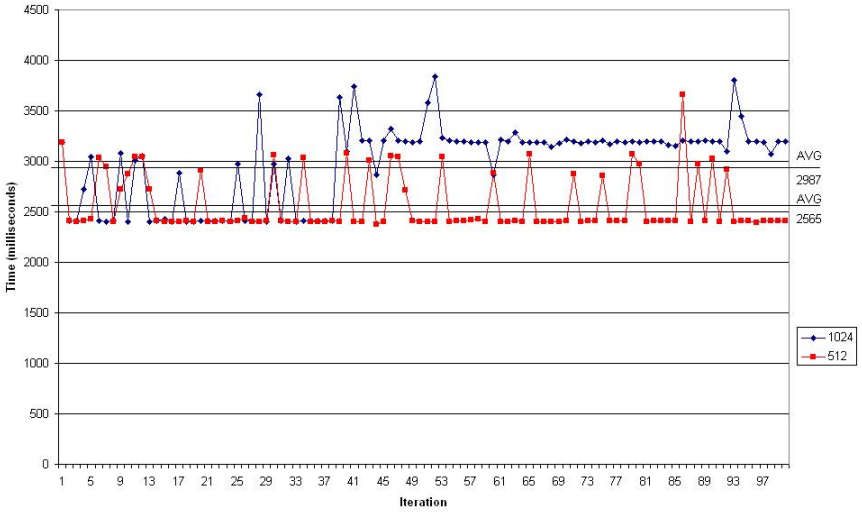

Figure 5.8: Timing measurements for the ÆTHER$_1$ binding process.

The average time required for a full binding with both devices having keys of 512 bits is 2.565 seconds (2,565 milliseconds) and 2.987 seconds (2,987 milliseconds) with keys of size 1,024 bits. The average required time in both cases is substantial, however the binding of devices in ÆTHER$_1$ is not a very frequent process and therefore this overhead does not introduce any usability problems for the human owner of the two devices involved.

5.3.2.2 Attribute Certificate Acquisition

In ÆTHER$_1$ the Attribute Certificate Acquisition Protocol (ACAP) takes place over the infrared channel established between the issuer and the subject devices; the complete process has been discussed in paragraph 4.4.3. The owner of the issuer device selects the "issue AC" operation for the appropriate attribute on her device which generates a random number R and sends it to the subject device with an $ACSYN$ message over the infrared interface. The subject device signs R with its private key and includes the signature as well as its public key to the $ACACK$ reply message. The issuer verifies the signature, builds the AC and sends it to the subject with an $ACCERT$ message. In our experiments for this protocol we have measured the required time from the point that the owner of the issuer device selects the appropriate operation, to the point that her device sends the $ACCERT$ message. The results are shown in Fig. 5.9.

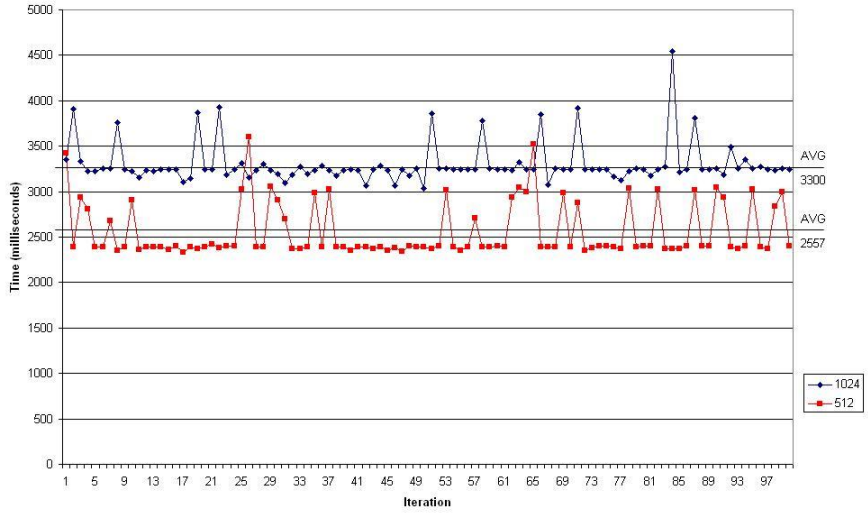

Figure 5.9: Timing measurements for the ÆTHER$_1$ ACAP.

According to the observed results the average time required for an issuer to complete the protocol is 2.557 seconds (2,557 milliseconds) when keys of 512 bits are used, and 3.3 seconds (3,300 milliseconds) with 1,024-bit keys. Although the average required time in both cases is significant, our tests demonstrated that it does not lead to usability problems. The users of the handheld devices can easily integrate the establishment of the required infrared channel and the issuing of an attribute certificate in their workflow.

5.3.2.3 Access Protocol

Our experiments for the ÆTHER$_1$ access protocol are based on the eager service discovery and access strategy, presented in paragraph 4.4.5.1, since it does not require human intervention for its completion as does the lazy strategy. We measured the time required from the point a device receives a public resource advertisement (a *PRA* message) from a service provider and sends a request (*REQ*) message for it, until the point it gets back a *REP* message that contains an authorization decision plus the provided service's interface, if the decision is positive, and enters this information into its CPV. We assume that there is an authorization policy for the provided service that specifies that a requester needs a single authority attribute to access the service. Moreover, that the requester has been issued an AC of the required type from a principal that is directly trusted by the provider to do so; i.e. the AC's issuer is listed in the *sources of authority* field of the AAS statement for the specific authority attribute that the provider has. The experiment was performed with the service provider's authorization cache both disabled and enabled in order to have demonstrative results (see Fig. 5.10).

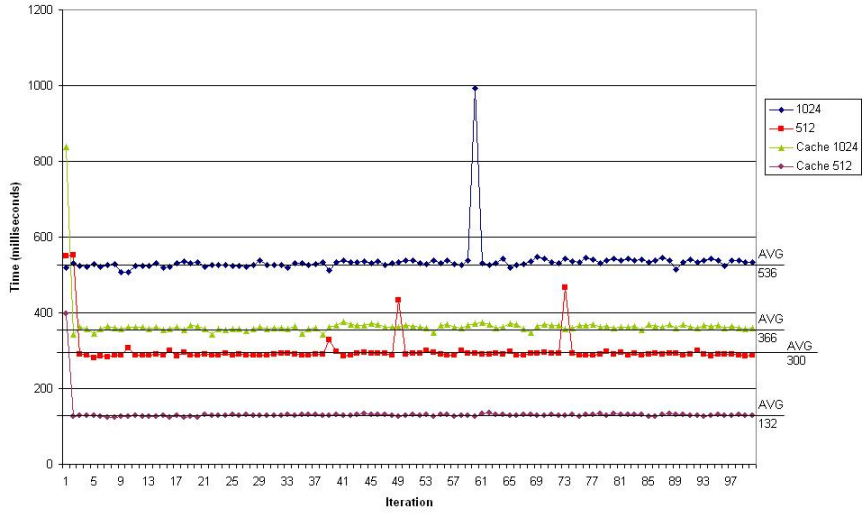

Figure 5.10: Timing measurements for the ÆTHER$_1$ access protocol.

When keys of 512 bits are used the average required time is 0.3 seconds (300 milliseconds). The utilization of the authorization cache decreases this to 0.132 seconds (132 milliseconds), a performance gain of 56%. With 1,024-bit keys the average required time

for the completion of the protocol is 0.536 seconds (536 milliseconds), and when we enable the authorization cache of the service provider this becomes 0.366 seconds (366 milliseconds). The decrease of the average required time is in the order of 31.7%. We believe that the timing results of both key sizes are realistic for a pervasive computing security system which relies on public key cryptography. Even in the eager service discovery and access strategy in which a device tries to access all services provided in an environment irrespectively of whether its user is interested in them, the time required for a successful completion of the protocol is half a second when 1,024-bit keys are used. This time is small enough not to be prohibitive.

5.3.2.4 Policy Evaluation

The inference engine of the ÆTHER$_1$ instantiation has been evaluated according to two metrics; the number of authority attributes that are required to reach an authorization decision, and the delegation depth that applies on principals that dynamically joined the AAS responsible for a required authority attribute. Both experiments have been performed with RSA keys of 512 and 1,024 bits. In the first one we have measured the time required for the inference engine to reach a positive decision when the number of the necessary authority attributes increases. Fig. 5.11 shows the results as averages of one hundred iterations for each number of authority attributes.

As expected the impact is more considerable when 1,024-bit keys are utilized. Specifically, with 512-bit keys the average increase of the required time when the number of authority attributes increases is 7.25 milliseconds, while with 1,024-bits the average increase is 15 milliseconds.

We have also evaluated the overhead introduced by the AAS's delegation depth parameter. In order to make this more clear consider the example presented in Fig. 5.12.

We have an AAS for the authority attribute *(group, family_member)* with a delegation depth value of 0 (denoting no restrictions on the depth of delegation) and a membership threshold value of 2. This AAS has as initial sources of authority the principals identified by the public keys A and B. In step 1 principal A issues an AC for the authority attribute *(group, family_member)* to principal C. This AC can be used by C to support the access

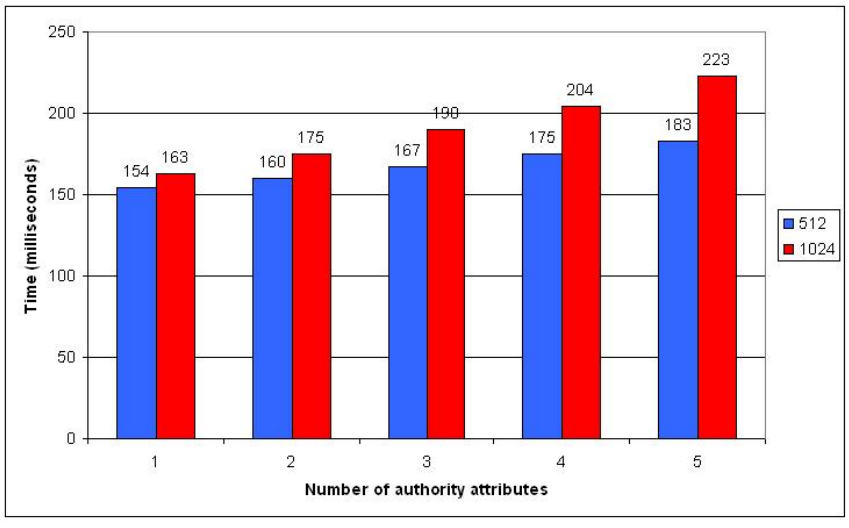

Figure 5.11: Impact of number of authority attributes on the ÆTHER$_1$ inference engine.

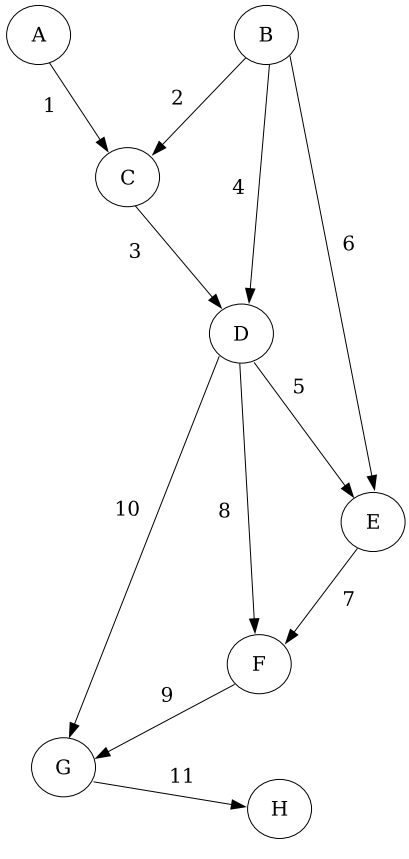

Figure 5.12: Delegation depth example.

requests it makes. When B issues an AC of the same type to C (step 2), C dynamically joins the *(group, family_member)* AAS (since we have an MTV of 2) and is now able to issue valid ACs of this type. It does so by issuing an AC to D (step 3) which makes an access request to a service provider. This is delegation depth 1 and the service provider must now verify three ACs, from steps 1, 2 and 3. The depth of delegation increases up to 5 where a service provider needs to perform 11 verification operations. Table 5.1 presents the required number of verifications at each delegation depth step.

Table 5.1: Delegation depth and required number of verifications for MTV 2.

Delegation depth	**Number of verifications**
1	2 + 1
2	4 + 1
3	6 + 1
4	8 + 1
5	10 + 1

We have performed the experiment one hundred times for each delegation depth step with RSA keys of 512 and 1,024 bits. In the first case we have found that the average increase in the required time to reach a positive authorization decision is 16.25 milliseconds, and in the second case 56 milliseconds. The specific results for each delegation depth step are shown in Fig. 5.13.

We believe that the observed performance results of the ÆTHER$_1$ inference engine illustrate the feasibility of using it on mobile constrained devices. Even when the delegation depth is 5, with an MTV of 2 and keys of 1,024 bits, the average time required to reach a decision is less than half a second (417 milliseconds) and therefore has minimal impact on the applications employed at a higher layer.

5.4 Memory Requirements

Table 5.2 presents the size in bytes of the compiled object code for each component of the two ÆTHER instantiations. It must be noted that the size of the object code greatly

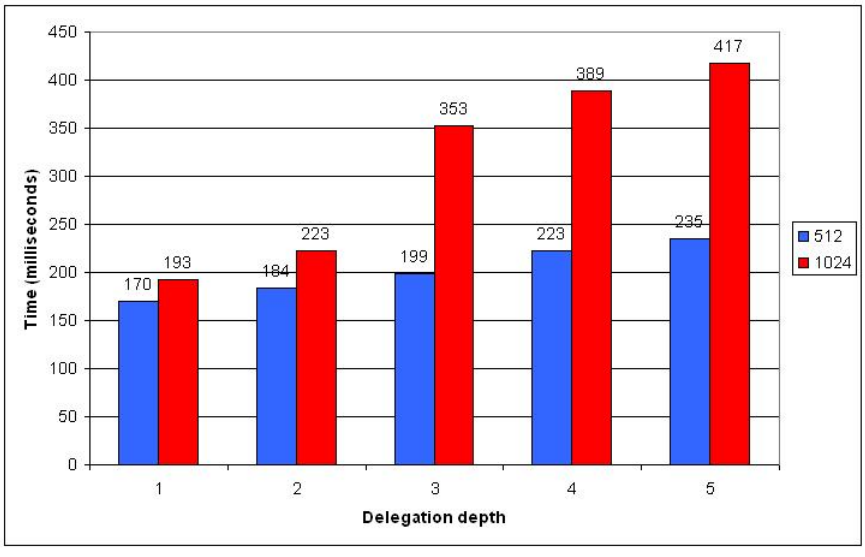

Figure 5.13: Impact of delegation depth on the ÆTHER$_1$ inference engine.

depends on the utilized compiler, the target hardware platform and possible code and compiler optimizations. Therefore, the given code sizes should not be strictly interpreted but only used to form a general idea of the memory requirements.

Table 5.2: Object code size for the ÆTHER components on the OMAP 1510 processor.

Component	ÆTHER$_0$ object code size (in bytes)	ÆTHER$_1$ object code size (in bytes)
Location-limited channel (infrared)	7,588	7,588
Authority renewal	3,662	4,547
Service discovery and access	9,940	16,188
Policy language parser	33,778	33,778
Cryptographic	9,304	9,304
Inference engine	6,458	7,252
Total	70, 730	78,657

The components that are common in both instantiations have, as expected, the same object code sizes. The total required code size for ÆTHER$_0$ is 69 KB (70,730 bytes) and for ÆTHER$_1$ is 77 KB (78,657 bytes). In both cases the policy language parser component, which is common in the two instantiations, takes up most of the code space. The second

largest component of both instantiations is the one that implements the service discovery and access protocols, requiring 9.7 KB (9,940 bytes) in ÆTHER$_0$ and 15.8 KB (16,188 bytes) in ÆTHER$_1$. We believe that these code sizes are small enough to allow both ÆTHER instantiations to be incorporated into most pervasive computing devices.

Another memory issue we have to examine is the storage space required for the ÆTHER policy data. Table 5.3 gives the size in bytes for typical instantiations of all the policy language statements used in ÆTHER. The largest statement used in ÆTHER$_0$ is the mutual binding statement, with a size of 293 bytes. Mutual bindings take place only between alpha-pad devices, which are more powerful in terms of computational and memory resources than low-end devices such as sensors, therefore this size is not restrictive. A sensor device that provides a single service would require enough memory space to store a binding, a positive authorization and an attribute assignment; a total of 688 bytes. However, this memory space is only required temporarily from the point an access request along with an attribute assignment arrives until the new association expires. The only policy statements that must always remain in the memory of the sensor are the binding and the positive authorization, which have a total size of 411 bytes.

In ÆTHER$_1$ the size of the policy statements depends primarily on the size of the utilized asymmetric key pairs. As an example consider that with 512-bit RSA key pairs attribute assignments (which in essence are attribute certificates) have a typical size of 645 bytes, while with 1,024-bit key pairs a typical size of 1,037 bytes.

5.5 Applications

In this section we will examine several application examples of the two instantiations of the general ÆTHER model. While some of these have been fully implemented and tested, others are beyond the means available to us and discussed only to analyze the expressiveness and the usefulness of our architecture.

5.5.1 Location Sensor

In this example we investigate the application of the ÆTHER$_0$ instantiation to secure a simple location sensor. While we refer to the sensor as a single device for practical purposes,

Table 5.3: Storage space required for the ÆTHER policy language statements.

Policy statement	ÆTHER$_0$ size (in bytes)	ÆTHER$_1$ with 512-bit keys (in bytes)	ÆTHER$_1$ with 1,024-bit keys (in bytes)
Binding	217	493	885
Mutual binding	293	-	-
Attribute assignment	277	645	1,037
Resource attribute assignment	223	581	973
Positive authorization	194	444	704
Permission resource attribute assignment	218	468	728
Composite positive authorization	229	503	763
Context attribute set	218	624	1,252
Context aggregator	248	498	758
Attribute mapping certificate	-	875	1,399
Authority attribute set	-	780	1,304

it can actually be a network of sensors reporting their findings to a server. This server can be an ÆTHER$_0$ alpha-pad that has bound every location sensor that participates in the network, collects the reported information and exposes it as a provided service to the local pervasive computing environment. The following positive authorization statement controls access to the location sensor service:

```
aether version: 0
type: positive authorization
issuer: "aabc125100cd462d1d7cf0aa50431616cb61"
resource: location_sensor
operation: get_list
requires: (@group == owner) || ($location == _location);
```

The principal specified in the *issuer* field is the one that bound the device that offers the location sensor service. The policy specifies that the membership list of the location sensor service (i.e. the list of people physically present in the current location) can be accessed by any principal that is in the same physical location as the service provider (that is the sensor itself). This is accomplished by specifying the value of the required *$location* attribute as a dynamic local context value that can be retrieved from the device's LCP. On the other hand, any principal that has been assigned the authority attribute *(group, owner)* can access the same information disregarding its physical location. The sensor device must also know that the principal responsible for maintaining the membership list of the current location is itself. We can accomplish this with the following CAS statement:

```
aether version: 0
type: context attribute set
issuer: "aabc125100cd462d1d7cf0aa50431616cb61"
attribute name: location
attribute value: _location
sources of authority: "1e56f3ddda76dbca3d83b9050eef0"
```

The hexadecimal string listed in the *sources of authority* field is the ÆTHER$_0$ identifier of the location sensor. Now the owner simply needs to change the value of the *$location* attribute in the device's LCP to a locally meaningful one, like for example *my_ living_ room*. When an access request is made to the location sensor, it checks to see if it has registered the requester as being physically present in the same location as itself, and allows access

to the service if it has. When the owner moves the location sensor from his living room to his office, he simply needs to change the value of the *$location* attribute in the device's LCP to a new one, for example *office_ 008*.

5.5.2 Pervasive Light Switch

We have implemented a simple pervasive light switch application in order to demonstrate some concepts of the ÆTHER$_1$ instantiation. The lamp is an incandescent light connected to a Phidget interface kit [Phi05]. The kit is connected to a laptop via a Universal Serial Bus (USB) interface and is controlled using the C++ Phidget API (see Fig. 5.14).

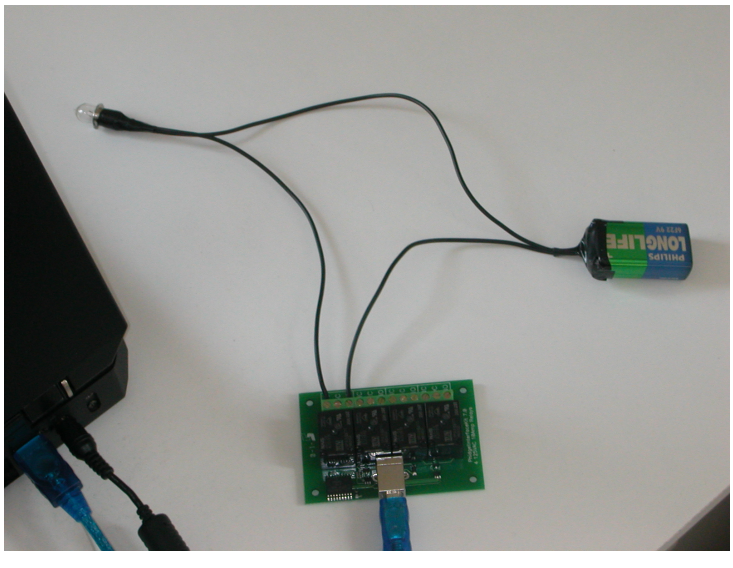

Figure 5.14: A pervasive light switch implemented using a phidget.

The laptop is the ÆTHER$_1$ principal that provides the light switch service to the local PAD. We consider this service to be an *output* service and we want to allow any member of the NTRG (our research group) or any visitor to our lab to access it,

```
aether version: 1
type: composite positive authorization
issuer: "3048024100cd462d1d7cf0aa50431616"
resource attribute name: service
resource attribute value: output
requires: (@ntrg_status == member) || (@ntrg_status == visitor);
signature: "af6a111dc085082bb103bd1ed351ac"
```

This policy has been issued to the laptop via a wireless interface by the principal that bound it in the past (the one identified in the *issuer* field). We also need to specify that the lamp service is considered an output service:

aether version: 1
type: resource attribute assignment
issuer: “3048024100cd462d1d7cf0aa50431616”
subject: “3048024100d1d645e46841f0bf1334ac”
resource attribute name: service
resource attribute value: output
signature: “23a668ed6437ab47c1168beab5165cf5”

Furthermore, we must define exactly what are the permissions associated with the lamp as an output service,

aether version: 1
type: permission resource attribute assignment
issuer: “3048024100cd462d1d7cf0aa50431616”
resource: light_switch
operation: change_state
resource attribute name: service
resource attribute value: output
signature: “904e9a7e58667712e55a390a0cc2f78”

The last two statements are example policy statements that can be provided by the lamp manufacturer as part of a default configuration state. Of course, in such a case they would be locally trusted and therefore they would not need to be signed. Finally, the lamp requires two AAS statements, for the *(ntrg_status, member)* and for the *(ntrg_status, visitor)* authority attributes. We just give here the second one:

aether version: 1
type: authority attribute set
issuer: “3048024100cd462d1d7cf0aa50431616”
attribute name: ntrg_status
attribute value: visitor
membership threshold value: 2
delegation depth: 1
sources of authority: “3048024100cd462d1d7cf0aa50431616”, “3048024100a9e3c6986a082d70e”
signature: “7f634b1fe3355a92d12d2bd6c10d94f”

Now any principal that has been given an AC of the authority attribute *(ntrg_status, visitor)* certified by either one of the two principals listed in the above AAS statement can change the state of the light switch. Moreover, since the AAS is dynamic with an MTV of 2, any NTRG visitor can introduce another visitor to our lab by issuing an *(ntrg_status, visitor)* AC provided she has been certified as a visitor by both sources of authority listed in the AAS statement.

5.5.3 Building Access Control

Current building access control systems rely on particularly static, inflexible and centrally managed technologies. The typical approach is to connect all door locks to a central server where access policy is regulated and register the identification numbers of specific swipe cards as authorized to open specific locks. Using ÆTHER$_1$ an organization or institution will be able to control access to physical spaces such as rooms and buildings in a dynamic manner, allowing disconnected operation while still maintaining sophisticated access control policies.

As an example consider a university (like Trinity College Dublin, or TCD) that wants to allow registered students to have access to particular areas. All locks controlling access to these areas can have the following positive authorization statement:

```
aether version: 1
type: positive authorization
issuer: "3048024100cd462d1d7cf0aa504316"
resource: lock
operation: unlock
requires: (@tcd_status == student);
signature: "811aab694f5436d26ca015ad52d9a"
```

Moreover, the locks need an AAS statement for defining which principals are trusted to certify the *(tcd_status, student)* authority attribute,

```
aether version: 1
type: authority attribute set
issuer: "3048024100cd462d1d7cf0aa504316"
attribute name: tcd_status
attribute value: student
membership threshold value: 3
```

delegation depth: 2
sources of authority: “3048024100a9e3c6986a0”, “3048024100c12fbaec493”, “3048024100ff69d2bf014”
signature: “d94f553340b62e7e58b89deb3ee”

The first principal in the *sources of authority* field is the university's buildings office, the second is the student records office, and the third one is the Provost's office. These three principals (since the MTV of the AAS is 3) can authorize another principal, like for example a specific university department, to issue valid *(tcd_status, student)* ACs. For example, the computer science department can give out such ACs to summer students without the need to contact any central management entity. This authority can be delegated even further (since the depth of allowed delegation is set to 2); the computer science department and two of the previous principals can authorize a specific research group to certify students with the *(tcd_status, student)* authority attribute.

In order to address emergency situations, like for example a fire, the *requires* field of the positive authorization statement installed at every lock can be changed to:

requires: (@tcd_status == student) || ($fire_alarm == on);

Finally, a CAS statement is needed in this case to define the principal responsible for providing the fire alarm service,

aether version: 1
type: context attribute set
issuer: “3048024100cd462d1d7cf0aa504316”
attribute name: fire_alarm
attribute value: on
sources of authority: “3048024100ab3ddca94cef74c”
signature: “95827c21bf513ad2eb09949a8”

5.5.4 Pervasive Car

Current automobiles have hundreds of embedded computers in order to provide services such as traction control, air conditioning, seat positioning, navigation and entertainment among others. As technology progresses with a particularly fast pace in this area, future automobiles will be full-fledged pervasive computing environments. By employing ÆTHER$_1$ the owner can specify, for example, that a parking driver is allowed to drive her car up to 20 miles per hour but not allowed to open the trunk.

The car manufacturer can predefine several authority and context attributes and ship them with every car; some of them are shown in Table 5.4.

Table 5.4: Authority and context attributes for a pervasive car.

Type of attribute	**Name and value**
Authority	(role, owner)
Authority	(role, driver)
Authority	(role, limited_driver)
Context	(role, passenger)

The first three authority attributes can be assigned different permissions related to the services of the car. For example, the *(role, owner)* attribute can have full access to every provided service, while the *(role, driver)* only allowed to start the engine, drive the car (with no speed limit), and access the navigation system. The *(role, passenger)* context attribute is used to allow access to all passenger services, such as air conditioning, seat positioning and entertainment. Its membership list is maintained by the sensors of the car, that are defined in a CAS statement to be the sources of authority for this context attribute. All these policy statements describe the default behavior of the car and can therefore be shipped by the manufacturer. As an example consider the following positive authorization statement:

```
aether version: 1
type: positive authorization
issuer: "3048024100ab3ddca94cef74c5"
resource: air_condition
operation: operate
requires: (@role == owner) || ($role, passenger);
signature: "8a793ae3d33aa81f18e206784"
```

Another possible application is an anti-theft service that periodically requests from the principal driving the car a specific set of credentials. This request can take place between the car's ignition (the service provider) and the ignition key used to start the engine (the service requester). If the ignition key does not provide the necessary credentials issued by a principal trusted by the ignition to do so (specified in an AAS statement), the car slows

down, stops and initiates the broadcasting of a "stolen" message. Such an authorization can be expressed as follows,

```
aether version: 1
type: positive authorization
issuer: "3048024100ab3ddca94cef74c5"
resource: engine
operation: keep_running
requires: (@role == owner) || (@role == driver) || (@role == limited_driver);
signature: "2d5077019a1fc2d11128df630c4"
```

5.5.5 Web Services

Although web services are not directly related to pervasive computing, we believe that the two domains have some similarities. For example, in both domains *a priori* knowledge of the complete set of participating entities and global centralized trust registers cannot be assumed. The WebÆTHER project, an M.Sc. dissertation supervised by the author, investigated the application of the ÆTHER$_1$ instantiation to this domain [Ste05].

WebÆTHER is a security management system that focuses on access control for web services. The approach of WebÆTHER to web services security is different than the WS-* standards series [OAS04], [WS-02], [WS-04]. It focuses on trust management and the ability to distribute policies between previously unknown entities by using the ÆTHER$_1$ decentralized architecture rather than the XML Key Management Specification (XKMS) [W3C05] where a traditional PKI is used. Attribute-based authorization is identical to ÆTHER$_1$ and authority domains are considered the servers that provide web services. Policies and authority attribute sets are specific for each web service and ACs are distributed to requesters of a service.

In WebÆTHER web services are installed as Common Gateway Interface (CGI)[3] scripts on an Apache web server. In order to implement WebÆTHER, the web server is configured to require X.509 identity certificates since HTTPS is used. CGI web services are designed to work with HTTPS and client-side authentication. The components of the implemented system are illustrated in Fig. 5.15.

[3]CGI is a standard that enables a client to request data from a program executed on a web server.

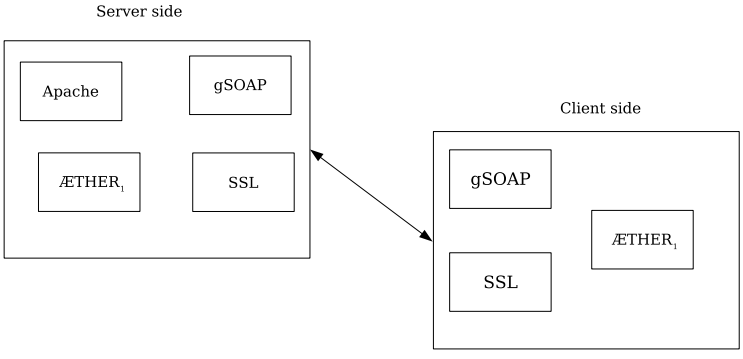

Figure 5.15: WebÆTHER system components.

The SOAP implementation provided by the gSOAP toolkit [gSO05] and the SSL implementation of the Apache web server form the proxy component that handles message transport. The Apache web server component provides the service in the form of a CGI in the server; the ÆTHER_1 component provides the trust management functionality in the server side and the handling of ACs in the client side. The X.509 identity certificates that the clients use during the SSL client authentication process must include the same public key as the one that is used in the *subject* field of the ÆTHER_1 ACs of the client principal.

Certificate acquisition is also implemented as a CGI web service. The administrator of the service first decides what ACs must be issued to a client. She then sends a randomly generated temporary URL of the service to the client. After the client connects to the service and retrieves the ACs, the specific URL is deleted.

For more information on WebÆTHER, as well as evaluation results, please see [Ste05].

Chapter 6

Conclusions and Future Work

As the vision of pervasive computing becomes reality an increasing number of personal mobile devices, fixed single function devices (like sensors) and household appliances will become interconnected, form networks, and offer their services to each other. The main enabling technologies behind the formation of such local pervasive networks will be wireless media like IEEE 802.11 WLAN and Bluetooth. Security and access control of the provided data and services will be a major enabling factor for the adoption of these new technologies. We, the research community, must address the fundamental security requirements that will allow the adoption of pervasive computing by a large audience. Security mechanisms should be incorporated from the design phase into our architectures in order to achieve robust systems that can withstand malicious participants. Moreover, we must always have the fundamental pervasive computing requirement in mind: unobtrusiveness. It is not realistic to expect human users to constantly be involved in the process of answering security configuration dialogs and managing the hundreds of devices that will exist in their environments. Thus, the simplification of the security management process and its graceful integration with the physical actions of the users becomes a central design issue. Context information, such as user location and current activity, can be utilized in order to facilitate simplified management without substantial distractions.

Furthermore, pervasive computing environments are inherently open and dynamic; participating devices are mobile, wearable, and in the near future implantable. As users roam among administrative domains the devices they carry must be able to interact in a discon-

nected and decentralized manner. Secure connections must be established spontaneously, without the need to access online central entities.

A detailed analysis of the existing traditional and pervasive security management systems focused on the above requirements revealed that there is a need for an authorization framework which is able to provide fine-grained access control capabilities, without depending on external centralized infrastructure. Moreover, such a framework must be able to support disconnected operation and gracefully integrate the human intervention required for the administrative tasks with the active physical task at hand in order to minimize distractions.

In this dissertation we have presented ÆTHER, our solution to the above problem. ÆTHER is a framework for pervasive computing authorization management that directly extends the traditional RBAC model for supporting decentralized administration, disconnected operation, context-awareness and follows an unobtrusive usage model. Based on this general framework we have instantiated two different systems. ÆTHER_0 has been designed to address the authorization requirements of small pervasive environments which consist of particularly constrained devices. It relies only on symmetric key cryptography by sacrificing the local decentralization requirement. In ÆTHER_0 all authority flows from alpha-pads, devices that directly represent particular users in the system. Our second instantiation, ÆTHER_1, is both globally and locally decentralized. However, this greater flexibility comes at greater computational requirements from the participating devices since we use public key cryptography and certificates for the required attribute assignments. In ÆTHER_1 the sets of principals that act as sources of authority for specific attributes are allowed to grow dynamically supporting the establishment of trust with unknown principals and authorizing their actions in the local domain. Our implementation and evaluation of the two ÆTHER instantiations demonstrated the feasibility of deploying and using them in pervasive computing environments. We have also investigated the expressiveness of our framework by using it to secure different applications, both futuristic and fully developed ones.

6.1 Summary of Contributions

The major contributions of this dissertation are the following:

- The analysis of the requirements of the problem of authorization management in pervasive computing. A detailed examination of traditional and pervasive computing security management systems that exist in the literature and identification of the reasons they fail to provide a complete solution to the problem.
- The formulation of a comprehensive threat model for the pervasive computing paradigm. This included an examination of the new and previously overlooked avenues of attack introduced by context-awareness.
- The development of an authorization management framework that satisfies the previously identified requirements. Specifically, it provides decentralized, disconnected, context-aware and unobtrusive security to pervasive computing environments.
- The extensions to the concept of location-limited channels to use them as part of users' physical actions to state their intent regarding authorization while being integrated gracefully with the primary task at hand.
- The enhancements to the KeyNote trust management system to facilitate its use in implementing our general framework.
- The instantiation of the general framework into two distinct systems that address the security, computational and user demands of different pervasive authority domains by employing different cryptographic mechanisms, design approaches and management models.
- The demonstration of the feasibility of the proposed framework and its two instantiations by implementing and evaluating them in a real-world pervasive computing environment consisting of modern handheld devices.

6.2 Future Work

During the design, implementation and evaluation of the ÆTHER framework we have identified several directions that future research on the subject can follow. These are briefly summarized here:

- The area of policy specification is one that requires further research. Although our current policy specification language is not overly complicated, it may still prove to be difficult for ordinary users to understand and use. The development of a high-level policy language can allow ÆTHER users to specify their security requirements in loosely structured natural language. This can then be translated by the system into the form expected by the parser component.

- Although we have performed a quantitative evaluation of the ÆTHER framework, we have not evaluated it from a qualitative perspective. Future work must focus on ensuring that our user-based authorization model and the integration of the establishment of location-limited channels with the primary physical actions of users conform to recognized usability guidelines, like the ones presented in [Nie94]. Such an analysis should take place in the actual environments that the users of the system perform their daily activities; offices, homes, restaurants, etc.

- The ÆTHER framework uses short expiration time periods for handling the problem of authority revocation. The fundamental problem of this approach is the selection of the expiration time period. Further research on this area can lead to a detailed risk management analysis of particular pervasive computing applications and ultimately to the recommendation of specific validity periods. These should be validated with both formal justification and practical experimentation.

- For the implementation of the ÆTHER framework we have relied on modern hand-held devices. Future work must focus on porting ÆTHER, and particularly the ÆTHER_0 instantiation, to constrained devices like 8-bit sensors. A thorough quantitative evaluation should follow to prove the feasibility of deploying the system on such platforms.

Furthermore, we have identified two general pervasive computing areas that future research should focus on. Pervasive computing is a relatively new, multidisciplinary paradigm for defining the future of the role that information processing devices will play in our lives. As such its goals are broad and ambitious. Human users and the interactions they have based on their social relationships must always be at the center of any pervasive computing system; indeed this goal effectively summarizes the vision of pervasiveness. However, all our current system design methodologies and guidelines fail to capture this essential requirement. The main reason is that although humans and their interactions are included in the design process, their behavior remains unpredictable and outside the control of system designers. Therefore, all efforts to design management systems, including security management systems such as ÆTHER, are problematic.

It is our belief that considerable research is required on two fields of pervasive computing; the reexamination of traditional design approaches regarding the unpredictability introduced by human users as system components, and the definition of clear and concise metrics that will allow us to compare the different proposed solutions. This work will allow us to both build and evaluate dependable pervasive computing security systems that fully integrate their intended users as system components.

Appendix A

The ÆTHER Policy Language

This appendix presents the full Backus-Naur Form (BNF) [Knu64] notation of the ÆTHER policy language.

<Statement>:: <AttributeAssignment> | <SignedAttributeAssignment> | <ResourceAttributeAssignment>
| <SignedResourceAttributeAssignment> | <BindingPolicy> | <SignedBindingPolicy>
| <MutualBinding> | <PositiveAuthorization> | <SignedPositiveAuthorization>
| <CompositePositiveAuthorization> | <SignedCompositePositiveAuthorization>
| <CompositeNegativeAuthorization> | <SignedCompositeNegativeAuthorization>
| <NegativeAuthorization> | <SignedNegativeAuthorization>
| <PermissionResourceAttributeAssignment> | <SignedPermissionResourceAttributeAssignment>
| <AuthorityAttributeSet> | <SignedAuthorityAttributeSet> | <ContextAttributeSet>
| <SignedContextAttributeSet> | <ContextAggregator> | <SignedContextAggregator>
| <AttributeMappingCertificate> | <SignedAttributeMappingCertificate> ;

<AttributeAssignment>:: <Version> <TypeAA> <Issuer> <Subject> <AttributeName> <AttributeValue>
<NotValidBefore> <NotValidAfter> <Renewable> ;

<SignedAttributeAssignment>:: <AttributeAssignment> <Signature> ;

<ResourceAttributeAssignment>:: <Version> <TypeRAA> <Issuer> <Subject>
<ResourceAttributeName> <ResourceAttributeValue> ;

<SignedResourceAttributeAssignment>:: <ResourceAttributeAssignment> <Signature> ;

<Binding>:: <Version> <TypeB> <Issuer> <Subject> ;

<BindingZero>:: <Binding> <SharedSecret> ;

<SignedBinding>:: <Binding> <Signature> ;

<MutualBinding>:: <Version> <TypeMB> <Issuer> <Subject> <NotValidBefore> <NotValidAfter>
<SharedSecret> ;

<PositiveAuthorization>:: <Version> <TypePA> <Issuer> <Resource> <Operation> <RequiresTag> <Requirements> ;

<SignedPositiveAuthorization>:: <PositiveAuthorization> <Signature> ;

<CompositePositiveAuthorization>:: <Version> <TypeCPA> <Issuer> <ResourceAttributeName> <ResourceAttributeValue> <RequiresTag> <Requirements> ;

<SignedCompositePositiveAuthorization>:: <CompositePositiveAuthorization> <Signature> ;

<CompositeNegativeAuthorization>:: <Version> <TypeCNA> <Issuer> <ResourceAttributeName> <ResourceAttributeValue> <RequiresTag> <Requirements> ;

<SignedCompositeNegativeAuthorization>:: <CompositeNegativeAuthorization> <Signature> ;

<NegativeAuthorization>:: <Version> <TypeNA> <Issuer> <Resource> <Operation> <RequiresTag> <Requirements> ;

<SignedNegativeAuthorization>:: <NegativeAuthorization> <Signature> ;

<PermissionResourceAttributeAssignment>:: <Version> <TypePRAA> <Issuer> <Resource> <Operation> <ResourceAttributeName> <ResourceAttributeValue> ;

<SignedPermissionResourceAttributeAssignment>:: <PermissionResourceAttributeAssignment> <Signature> ;

<AuthorityAttributeSet>:: <Version> <TypeAAS> <Issuer> <AttributeName> <AttributeValue> <Threshold> <Delegation> <AuthorityTag> <Authorities> ;

<SignedAuthorityAttributeSet>:: <AuthorityAttributeSet> <Signature> ;

<ContextAttributeSet>:: <Version> <TypeCAS> <Issuer> <AttributeName> <AttributeValue> <AuthorityTag> <Authorities> ;

<SignedContextAttributeSet>:: <ContextAttributeSet> <Signature> ;

<ContextAggregator>:: <Version> <TypeCA> <Issuer> <InputTag> <Requirements> <OutputName> <OutputValue> ;

<SignedContextAggregator>:: <ContextAggregator> <Signature> ;

<AttributeMappingCertificate>:: <Version> <TypeAMC> <Issuer> <AttributeName> <AttributeValue> <SubjectAttributeName> <SubjectAttributeValue> <SubjectAuthorityTag> <SubjectAuthorities> <NotValidBefore> <NotValidAfter> ;

<SignedAttributeMappingCertificate>:: <AttributeMappingCertificate> <Signature> ;

<AttributeName>:: “attribute” “name” “:” <String> | “attribute” “name” “:” “name” ;

<AttributeValue>:: “attribute” “value” “:” <String> | “attribute” “value” “:” <Number> | “attribute” “value” “:” “<user-defined>” ;

<Authorities>:: <Authority> | <Authorities> “,” <Authority> ;

<AuthorityTag>:: “sources” “of” “authority” “:” ;

<Delegation>:: “delegation” “depth” “:” <Number> ;

<InputTag>:: “input” “:” ;

<Issuer>:: “issuer” “:” DQUOTE <String> DQUOTE ;

<NotValidAfter>:: “not” “valid” “after” “:” <Date> ;

<NotValidBefore>:: “not” “valid” “before” “:” <Date> ;

<Operation>:: “operation” “:” <String> ;

<OutputName>:: “output” “context” “attribute” “name” “:” <String> | “output” “context” “attribute” “name” “:” “name” ;

<OutputValue>:: “output” “context” “attribute” “value” “:” <String> | “output” “context” “attribute” “value” “:” <Number> ;

<Renewable>:: “renewable” “:” <Number> ;

<Requirements>:: <Requirement> | <Requirements> “&&” <Requirement>
| <Requirements> “||” <Requirement> ;

<RequiresTag>:: “requires” “:” ;

<Resource>:: “resource” “:” <String> ;

<ResourceAttributeName>:: “resource” “attribute” “name” “:” <String>
| “resource” “attribute” “name” “:” “name” ;

<ResourceAttributeValue>:: “resource” “attribute” “value” “:” <String>
| “resource” “attribute” “value” “:” “output”
| “resource” “attribute” “value” “:” “input”
| “resource” “attribute” “value” “:” <Number>
| “resource” “attribute” “value” “:” “<user-defined>” ;

<SharedSecret>:: “shared” “secret” “:” DQUOTE <String> DQUOTE ;

<Signature>:: “signature” “:” DQUOTE <String> DQUOTE ;

<Subject>:: “subject” “:” DQUOTE <String> DQUOTE ;

<SubjectAttributeName>:: “subject” “attribute” “name” “:” <String> ;

<SubjectAttributeValue>:: “subject” “attribute” “value” “:” <String>
| “subject” “attribute” “value” “:” <Number>
| “subject” “attribute” “value” “:” “<user-defined>” ;

<SubjectAuthorities>:: <SubjectAuthority> | <SubjectAuthorities> “,” <SubjectAuthority> ;

<SubjectAuthorityTag>:: “subject” “sources” “of” “authority” “:” ;

<Threshold>:: “membership” “threshold” “value” “:” <Number> ;

<TypeAAS>:: “type” “:” “authority” “attribute” “set” ;

<TypeAA>:: “type” “:” “attribute” “assignment” ;

<TypeAMC>:: “type” “:” “attribute” “mapping” “certificate” ;

<TypeB>:: “type” “:” “binding” ;

<TypeCA>:: “type” “:” “context” “aggregator” ;

<TypeCAS>:: “type” “:” “context” “attribute” “set” ;

<TypeCNA>:: “type” “:” “composite” “negative” “authorization” ;

<TypeCPA>:: “type” “:” “composite” “positive” “authorization” ;

<TypeMB>:: “type” “:” “mutual” “binding” ;

<TypeNA>:: “type” “:” “negative” “authorization” ;

<TypePA>:: “type” “:” “positive” “authorization” ;

<TypePRAA>:: “type” “:” “permission” “resource” “attribute” “assignment” ;

<TypeRAA>:: “type” “:” “resource” “attribute” “assignment” ;

<Version>:: “aether” “version” “:” <Number> ;

<Authority>:: DQUOTE <String> DQUOTE ;

<Requirement>:: <String> “<” <Number> | <String> “>” <Number>
| <String> “<=” <Number> | <String> “>=” <Number>
| <String> “==” <String> | <String> “==” <Number>
| <String> “!=” <String> | <String> “!=” <Number> ;

<SubjectAuthority>:: DQUOTE <String> DQUOTE ;

<Attributes>:: <Attribute> | <Attributes> “&&” <Attribute> | <Attributes> “||” <Attribute> ;

<Attribute>:: “(“ <String> “,” <String> “)” | “(“ <String> “,” <Number> “)” ;

<String>:: { [@$%_*]*[a-zA-Z0-9_]+ } ;

<Number>:: { [0-9]+ } ;

<Date>:: { [2]([0-9]{3})“/”[0-9]{1,2}“/”[0-9]{1,2}“-”[0-9]{1,2}“:”[0-5]([0-9]) } ;

Appendix B

Performance Analysis of Cryptographic Primitives

The use of mobile computing devices (e.g. handhelds, palmtops, mobile phones) has increased over the years, particularly during the last decade. Personal Digital Assistants (PDAs) started initially as devices to store personal information. As they have grown more compact with more powerful CPUs, they have evolved to support more advanced communications applications that have traditionally been the domain of workstations. At the same time there have been significant changes in the way business is done with the introduction of electronic commerce endeavors through the Internet. Electronic commerce involves the use of strong cryptographic functions and protocols in order to provide adequate security services for payment transactions. These functions can be easily afforded by fixed workstations, but the literature [DB99], [GG01] would suggest that on mobile devices are slow and expensive due to constrained processors, limited memory and battery life. The latest generations of mobile devices are equipped with much faster CPUs, which facilitate the use of strong cryptographic functions for the construction of security-related protocols.

In this appendix we present timing measurements for low-level cryptographic primitives such as symmetric and asymmetric cryptography operations, as well as message digests. The hardware platform as well as the experiment parameters we use are the same as the ones presented in paragraph 5.3. Fig. B.1 shows the time in milliseconds that is required

to perform 100,000 operations for each investigated algorithm.

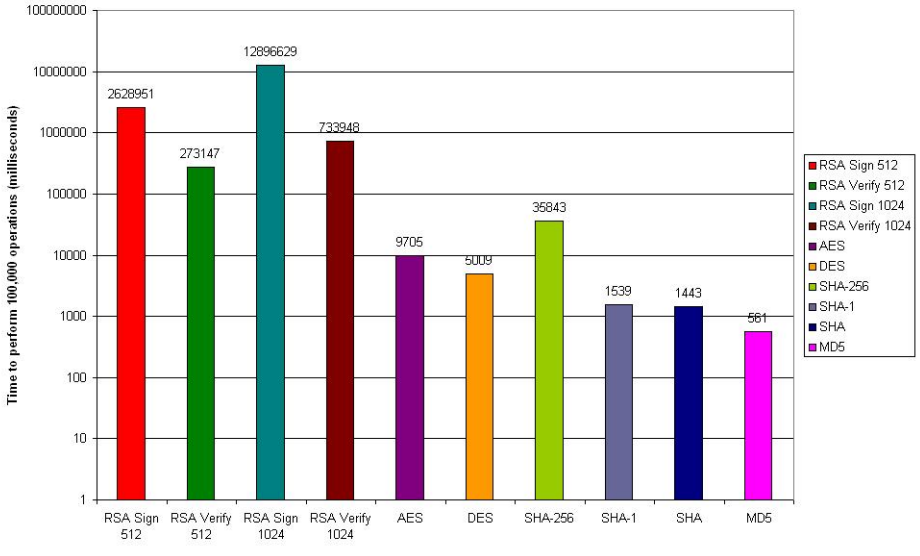

Figure B.1: Timing measurements of cryptographic primitives on an iPAQ H6340.

For all the RSA operations we use an input message of 64 bytes and the SHA-1 message digest algorithm for hashing the input before signing it and verifying its signature. For the symmetric algorithms (DES and AES) we again use a 64-byte message as input, with a 56-bit key for DES and a 256-bit key for AES. The examined message digests (SHA-256, SHA-1, SHA and MD5) were also given a 64-byte input message to process. Fig. B.2 presents the number of operations that each algorithm is able to perform per second.

Our experiments show that, for example, the RSA algorithm using 1,024-bit keys can perform 8 signature generation operations per second while the MD5 algorithm can perform 178,253 hashing operations per second; a four orders of magnitude difference. We have also investigated the time required for generating new RSA key pairs (see Table B.1). On average 2.48 seconds are required to generate a 512-bit RSA key pair, and 10.76 seconds for the generation of a 1,024-bit key pair.

These results demonstrate that currently available handheld devices can form the foundation of secure ubiquitous computing environments since they can facilitate the use of strong cryptographic functions.

The work presented in this appendix complements previous attempts to implement

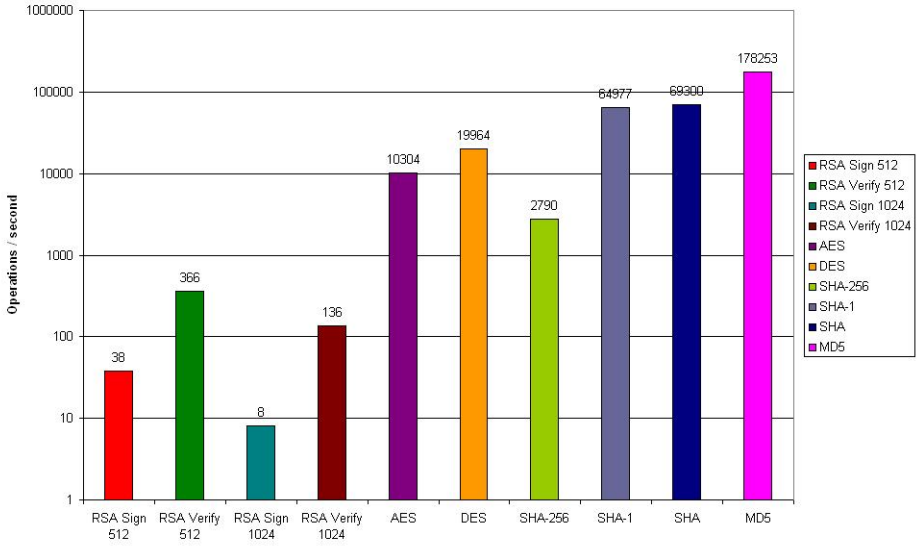

Figure B.2: Cryptographic operations per second on an iPAQ H6340.

Table B.1: RSA key pair generation on an iPAQ H6340.

Operation	**Time required for 1,000 iterations**
512 bits RSA key pair generation	2,471.614 seconds (2,471,614 milliseconds)
1,024 bits RSA key pair generation	10,760.783 seconds (10,760,783 milliseconds)

and use cryptographic protocols on mobile handheld devices. Although a comprehensive comparison between our work and previous similar attempts cannot be accomplished due to different hardware and software parameters, there are some useful comments that can be noted regarding advances in handheld computing technology. In [GG01] the authors examine RSA operations on a 20 MHz Palm CPU with keys of sizes 768 and 1,024 bits using the Java 2 Micro-Edition software platform. Their results indicate that they require 0.5-1.5 seconds. RSA operations were also investigated in the context of electronic commerce through the use of handheld devices [DB99]. The platform in this case was a PalmPilot Professional with a Motorola DragonBall chip at 16 MHz, running the PalmPilot port of the SSLeay cryptographic library. The observed results for RSA operations with 512 bits key pairs were 3.4 minutes for key generation, 7 seconds (7,028 milliseconds) for signing and 1.4 seconds (1,376 milliseconds) for verification. As it is obvious the order of time required for RSA operations has been reduced from seconds to milliseconds. However, such a comparison is only useful to demonstrate advances in handheld computing technology since the hardware platform we have used is more fitting to perform CPU intensive cryptographic operations.

Bibliography

[Aba02] M. Abadi. Private Authentication. In *Proceedings of 2002 Workshop on Privacy Enhancing Technologies (PET'02)*, pages 27–40, 2002.

[AF99] C. Adams and S. Farrell. Internet X.509 Public Key Infrastructure, Certificate Management Protocols. RFC 2510, 1999.

[AG00] N. Asokan and P. Ginzboorg. Key-agreement in Ad hoc Networks. *Computer Communications*, 17(23):1627–1637, 2000.

[AGRS05] F. Adelstein, S.K.S. Gupta, G.G. Richard, and L. Schwiebert. *Fundamentals of Mobile and Pervasive Computing.* McGraw-Hill, 2005.

[AJLS97] M.D. Addlesee, A. Jones, F. Livesey, and F. Samaria. The ORL Active Floor. *IEEE Personal Communications*, 4(5):35–41, 1997.

[AK96] R. Anderson and M. Kuhn. Tamper Resistance - A Cautionary Note. In *Proceedings of 2nd USENIX Workshop on Electronic Commerce*, 1996.

[AK97] R. Anderson and M. Kuhn. Low Cost Attacks on Tamper Resistant Devices. In *Proceedings of 5th Internation Workshop on Security Protocols*, pages 125–136, 1997.

[AMRCM03] J. Al-Muhtadi, A. Ranganathan, R. Campbell, and D. Mickunas. Cerberus: a Context-aware Security Scheme for Smart Spaces. In *Proceedings of 1st IEEE International Conference on Pervasive Computing and Communications (PERCOM'03)*, pages 489–496, 2003.

[And01] R. Anderson. *Security Engineering.* John Wiley & Sons, 2001.

[ASW+99] K. Arnold, R.W. Scheifler, J. Waldo, A. Wollrath, and B. O'Sullivan. *The Jini Specification.* Addison-Wesley, 1999.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[BC04] E. Biham and R. Chen. Near-collisions of SHA-0. Cryptology ePrint Archive, Report 2004/146, http://eprint.iacr.org/2004/146/, 2004.

[BCM+02] M. Burnside, D. Clarke, T. Mills, S. Devadas, and R.L. Rivest. Proxy-based Security Protocols in Networked Mobile Devices. In *Proceedings of 17th ACM Symposium on Applied Computing (SAC'02)*, 2002.

[BF01] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. In *Proceedings of 21st Annual International Cryptology Conference (CRYPTO'01)*, pages 213–229, 2001.

[BFK99] M. Blaze, J. Feigenbaum, and A.D. Keromytis. The KeyNote Trust Management System (Version 2). RFC 2704, 1999.

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of 1996 IEEE Symposium on Security and Privacy*, pages 164–173, 1996.

[BGW01] N. Borisov, I. Goldberg, and D. Wagner. Intercepting Mobile Communications: the Insecurity of 802.11. In *Proceedings of 7th Annual International Conference on Mobile Computing and Networking (MOBICOM'01)*, pages 180–189, 2001.

[Blu03] Bluetooth Special Interest Group. v.1.2. Core Specification of the Bluetooth System. http://www.bluetooth.org/spec/, 2003.

[BM03] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer, 2003.

[BMK+00] B. Brumitt, B. Meyers, J. Krumm, A. Kern, and S. Shafer. EasyLiving: Technologies for Intelligent Environments. In *Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, pages 12–29, 2000.

[BRH94] F. Bennett, T. Richardson, and A. Harter. Teleporting - Making Applications Mobile. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 82–84, 1994.

[BS02] D. Balfanz and D.K. Smetters. Talking to Strangers: Authentication in Ad hoc Wireless Networks. In *Proceedings of 9th Network and Distributed System Security Symposium (NDSS'02)*, 2002.

[CBH03] S. Capkun, L. Buttyan, and J.-P. Hubaux. Self-organized Public Key Management for Mobile Ad hoc Networks. *IEEE Transactions on Mobile Computing*, 1(2):52–64, 2003.

[CD00] D. Caswell and P. Debaty. Creating Web Representations for Places. In *Proceedings of 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K)*, pages 114–126, 2000.

[CEE+01] D. Clarke, J.-E. Elien, C. Ellison, M. Fredette, A. Morcos, and R.L. Rivest. Certificate Chain Discovery in SPKI/SDSI. *Journal of Computer Security*, 9(4):285–322, 2001.

[CFJ03] H. Chen, T. Finin, and A. Joshi. An Ontology for Context-aware Pervasive Computing Environments. *Knowledge Engineering Review*, 18(3):197–208, 2003.

[CGRZ03] S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. The Attacker in Ubiquitous Computing Environments: Formalising the Threat Model. In *Proceedings of 1st International Workshop on Formal Aspects in Security and Trust*, pages 83–97, 2003.

[CH96] B. Christianson and W.S. Harbison. Why Isn't Trust Transitive? In *Proceedings of 3rd International Workshop on Security Protocols*, pages 171–176, 1996.

[CK00] G. Chen and D. Kotz. A Survey of Context-aware Computing Research. Technical Report TR2000-381, Dartmouth College, Department of Computer Science, 2000.

[CLS+01] M.J. Covington, W. Long, S. Srinivasan, A. Dey, M. Ahamad, and G. Abowd. Securing Context-aware Applications using Environment Roles. In *Proceedings of 6th ACM Symposium on Access Control Models and Technologies (SACMAT'01)*, pages 10–20, 2001.

[CM02] C. Castelluccia and G. Montenegro. Protecting AODV Against Impersonation Attacks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 6(3):108–109, 2002.

[CMA00] M.J. Covington, M.J. Moyer, and M. Ahamad. Generalized Role-Based Access Control for Securing Future Applications. In *Proceedings of 23rd National Information Systems Security Conference (NISSC'00)*, 2000.

[CO02] D.W. Chadwick and A. Otenko. The PERMIS X.509 Role-Based Privilege Management Infrastructure. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pages 135–140, 2002.

[DA99] A.K. Dey and G.D. Abowd. Towards a Better Understanding of Context and Context-awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, College of Computing, 1999.

[DB99] N. Daswani and D. Boneh. Experimenting with Electronic Commerce on the PalmPilot. In *Proceedings of 3rd International Conference on Financial Cryptography (FC'99)*, pages 1–16, 1999.

[Der99] M.L. Dertouzos. The Future of Computing. *Scientific American*, 282(3):52–63, 1999.

[DH76] W. Diffie and M.E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[Don03] M. Donner. Toward a Security Ontology. *IEEE Security & Privacy*, 1(3):6–7, 2003.

[DR02] J. Daemen and V. Rijmen. *The Design of Rijndael, AES - The Advanced Encryption Standard.* Springer-Verlag, 2002.

[Dro97] R. Droms. Dynamic Host Configuration Protocol (DHCP). RFC 2131, 1997.

[EFL+99] C. Ellison, B. Frantz, B. Lampson, R.L. Rivest, B. Thomas, and T Ylonen. SPKI Certificate Theory. RFC 2693, 1999.

[FGBZ03] X. Fu, B. Graham, R. Bettati, and W. Zhao. Active Traffic Analysis Attacks and Countermeasures. In *Proceedings of 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03)*, pages 31–39, 2003.

[FH02] S. Farrell and R. Housley. An Internet Attribute Certificate Profile for Authorization. RFC 3281, 2002.

[FK92] D.F. Ferraiolo and D.R. Kuhn. Role-Based Access Controls. In *Proceedings of NIST-NSA National Computer Security Conference*, pages 554–563, 1992.

[FKC03] D.F. Ferraiolo, D.R. Kuhn, and R. Chandramouli. *Role-Based Access Control.* Artech House, 2003.

[FL01] S.R. Fluhrer and S. Lucks. Analysis of the E0 Encryption System. In *Proceedings of 8th Workshop on Selected Areas in Cryptography*, pages 38–48, 2001.

[For94] W. Ford. *Computer Communications Security - Principles, Standard Protocols and Techniques.* Prentice Hall, 1994.

[FSG+01] D. Ferraiolo, R. Sandhu, S. Gavrila, D.R. Kuhn, and R. Chandramouli. Proposed NIST Standard for Role-Based Access Control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[GG01] V. Gupta and S. Gupta. Securing the Wireless Internet. *IEEE Communications*, 39(12):68–74, 2001.

[Gli84] V.D. Gligor. A Note on Denial-of-Service in Operating Systems. *IEEE Transactions on Software Engineering*, SE-10(3):320–324, 1984.

[GPVD99] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol. RFC 2608, 1999.

[gSO05] gSOAP. SOAP C++ Web Services. http://gsoap2.sourceforge.net/, 2005.

[GWF+99] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring: WebDAV. RFC 2518, 1999.

[Hel05] S. Helal. Programming Pervasive Spaces. *IEEE Pervasive Computing*, 4(1):84–87, 2005.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile. RFC 2459, 1999.

[HMM+00] A. Herzberg, Y. Mass, J. Mihaeli, D. Naor, and Y. Ravid. Access Control Meets Public Key Infrastructure, or: Assigning Roles to Strangers. In *Proceedings of 2000 IEEE Symposium on Security and Privacy*, pages 2–14, 2000.

[HN99] M. Hermelin and K. Nyberg. Correlation Properties of the Bluetooth Combiner Generator. In *Proceedings of 2nd International Conference on Information Security and Cryptology*, pages 17–29, 1999.

[Hop99] A. Hopper. Sentient Computing. The Royal Society Clifford Paterson Lecture, ftp://ftp.uk.research.att.com/pub/docs/att/tr.1999.12.pdf, 1999.

[IEE97] IEEE Computer Society LAN/MAN Standards Committee. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11-1997, 1997.

[IPA05] HP iPAQ H6340. http://www.ipaqchoice.com/, 2005.

[ISO84] ISO. Information Processing Systems - Open Systems Interconnection (OSI) - Basic Reference Model. ISO-7498, 1984.

[ISO01] ISO/ITU-T Recommendation X.509. The Directory: Authentication Framework, 2001.

[JMB01] D.B. Johnson, D.A. Maltz, and J. Broch. *Ad hoc Networking*, chapter DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad hoc Networks, pages 139–172. Addison-Wesley, 2001.

[KA98] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[KB01] T. Kindberg and J. Barton. The Cooltown User Experience. Technical Report HPL-2001-22, Hewlett-Packard Labs, 2001.

[KBC97] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, 1997.

[Kee01] D. Keely. A Security Strategy for Mobile E-business. Technical Report GSOEE213, IBM Global Services, 2001.

[KF02] T. Kindberg and A. Fox. System Software for Ubiquitous Computing. *IEEE Pervasive Computing*, 1(1):70–81, 2002.

[KN93] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, 1993.

[Knu64] D.E. Knuth. Backus Normal Form vs. Backus-Naur Form. *Communications of the ACM*, 7(12):735–736, 1964.

[Kra01] H. Krawczyk. The Order of Encryption and Authentication for Protecting Communications (or: How Secure is SSL?). In *Proceedings of 21st Annual International Cryptology Conference (CRYPTO'01)*, 2001.

[KZL+01] J. Kong, P. Zerfos, H. Luo, S. Lu, and L. Zhang. Providing Robust and Ubiquitous Security Support for Mobile Ad hoc Networks. In *Proceedings of*

9th IEEE International Conference on Network Protocols (ICNP'01), pages 251–260, 2001.

[Lam74] B. Lampson. Protection. *ACM Operation Systems Review*, 8(1):18–24, 1974.

[Lan01] M. Langheinrich. Privacy by Design - Principles of Privacy-Aware Ubiquitous Systems. In *Proceedings of 3rd International Conference on Ubiquitous Computing (UBICOMP'01)*, pages 273–291, 2001.

[LDH04] Y.W. Law, J. Doumen, and P. Hartel. Benchmarking Block Ciphers for Wireless Sensor Networks. In *Proceedings of 2004 IEEE International Conference on Mobile Ad hoc and Sensor Systems*, pages 447–456, 2004.

[LMB92] J. Levine, T. Mason, and D. Brown. *Lex & Yacc, Second Edition.* O'Reilly, 1992.

[LN99] J. Linn and M. Nystrom. Attribute Certification: an Enabling Technology for Delegation and Role-Based Controls in Distributed Environments. In *Proceedings of 4th ACM Workshop on Role-Based Access Control*, pages 121–130, 1999.

[LS99] E. Lupu and M. Sloman. Conflicts in Policy-Based Distributed Systems Management. *IEEE Transactions on Software Engineering*, 25(6):852–869, 1999.

[LTZV04] B. LaPlant, S. Trewin, G. Zimmermann, and G. Vanderheiden. The Universal Remote Console: a Universal Access Bus for Pervasive Computing. *IEEE Pervasive Computing*, 1(3):76–80, 2004.

[LV04] Y. Lu and S. Vaudenay. Faster Correlation Attack on Bluetooth Keystream Generator E0. In *Proceedings of 24th Annual International Cryptology Conference (CRYPTO'04)*, pages 407–425, 2004.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560, 1999.

[MC04] D. Mundy and D.W. Chadwick. An XML Alternative for Performance and Security: ASN.1. *IEEE IT Professional*, 6(1):30–36, 2004.

[MESW01] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model - Version 1 Specification. RFC 3060, 2001.

[Mit04] C.J. Mitchell, editor. *Security for Mobility.* IEE, 2004.

[MIT05] MIT. MIT Project Oxygen. http://oxygen.lcs.mit.edu/, 2005.

[MJ98] P. McDaniel and S. Jamin. Windowed Key Revocation in Public Key Infrastructures. Technical Report CSE-TR-376-98, University of Michigan, Electrical Engineering and Computer Science Department, 1998.

[MMYH02] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino. Impact of Artificial "Gummy" Fingers on Fingerprint Systems. In *Proceedings of SPIE Vol. #4677, Optical Security and Counterfeit Deterrence Techniques IV*, pages 275–289, 2002.

[Moc87] P. Mockapetris. Domain Names: Implementation and Specification. RFC 1035, 1987.

[MR00] P. McDaniel and A. Rubin. A Response to 'Can We Eliminate Certificate Revocation Lists?'. In *Proceedings of 4th International Conference on Financial Cryptography (FC'00)*, 2000.

[MSW03] K.D. Mitnick, W.L. Simon, and S. Wozniak. *The Art of Deception: Controlling the Human Element of Security.* John Wiley & Sons, 2003.

[Nie94] J. Nielsen. *Usability Engineering.* Academic Press, Inc., 1994.

[NS78] R.M. Needham and M.D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[OAS04] OASIS Standard. Web Services Security: SOAP Message Security 1.0 (WS-Security 2004). http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf, 2004.

[OAS05] OASIS XACML Technical Committee. OASIS eXtensible Access Control Markup Language 2.0, Core Specification. http://www.oasis-open.org/committees/xacml/, 2005.

[OD01] D. O'Mahony and L. Doyle. *Mobile Computing: Implementing Pervasive Information and Communication Technologies*, chapter An Adaptable Node Architecture for Future Wireless Networks. Kluwer Publishing, 2001.

[OR87] D. Otway and O. Rees. Efficient and Timely Mutual Authentication. *Operating Systems Review*, 21(1):8–10, 1987.

[OR01] G. O'Shea and M. Roe. Child-proof Authentication for MIPv6 (CAM). *ACM SIGCOMM Computer Communication Review*, 31(2):4–8, 2001.

[Pas98] J. Pascoe. Adding Generic Contextual Capabilties to Wearable Computing. In *Proceedings of 2nd International Symposium on Wearable Computing (ISWC'98)*, pages 92–99, 1998.

[PCB00] N.B. Priyantha, A. Chakraborty, and H. Balakrishnan. The Cricket Location-Support System. In *Proceedings of 6th Annual International Conference on Mobile Computing and Networking (MOBICOM'00)*, pages 32–43, 2000.

[Phi05] Phidgets Inc. Physical Widgets (Phidgets). http://www.phidgets.com/, 2005.

[PLF+01] S.R. Ponnekanti, B. Lee, A. Fox, P. Hanrahan, and T. Winograd. ICrafter: a Service Framework for Ubiquitous Computing Environments. In *Proceedings of 3rd International Conference on Ubiquitous Computing (UBICOMP'01)*, pages 56–75, 2001.

[PR99] C.E. Perkins and E.M. Royer. Ad hoc On-Demand Distance Vector Routing. In *Proceedings of 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[PRM99] J. Pascoe, N. Ryan, and D. Morse. Issues in Developing Context-aware Computing. In *Proceedings of 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99)*, pages 208–221, 1999.

[PS00] J.S. Park and R. Sandhu. Binding Identities and Attributes using Digitally Signed Certificates. In *Proceedings of 16th Annual Computer Security Applications Conference*, pages 120–127, 2000.

[Res00] E. Rescorla. *SSL and TLS - Designing and Building Secure Systems*. Addison-Wesley, 2000.

[RHC+02] M. Roman, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt. Gaia: a Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, 2002.

[Riv98] R.L. Rivest. Can We Eliminate Certificate Revocation Lists? In *Proceedings of 2nd International Conference on Financial Cryptography (FC'98)*, pages 178–183, 1998.

[RL96] R.L. Rivest and B. Lampson. SDSI - A Simple Distributed Security Infrastructure. http://theory.lcs.mit.edu/cis/sdsi.html, 1996.

[RSA78] R.L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[SA00] F. Stajano and R. Anderson. The Resurrecting Duckling: Security Issues for Ad hoc Wireless Networks. In *Proceedings of 7th International Workshop on Security Protocols*, pages 172–182, 2000.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware Computing Applications. In *Proceedings of IEEE Workshop on Mobile Computing Systems and Applications*, pages 85–90, 1994.

[SCFY96] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[Sch99] B. Schneier. Attack Trees. *Dr. Dobb's Journal*, pages 21–29, 1999.

[SDA99] D. Salber, A.K. Dey, and G.D. Abowd. The Context Toolkit: Aiding the Development of Context-enabled Applications. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, pages 434–441, 1999.

[SSL05] OpenSSL Open Source Cryptographic Toolkit. http://www.openssl.org/, 2005.

[Sta01] F. Stajano. The Resurrecting Duckling - What Next? In *Proceedings of 8th International Workshop on Security Protocols*, pages 204–214, 2001.

[Sta02] F. Stajano. *Security for Ubiquitous Computing*. John Wiley & Sons, 2002.

[Ste05] P. Stefas. Decentralized Authorization for Web Services. Master thesis, University of Dublin, Trinity College, 2005.

[SW05] Y. Shaked and A. Wool. Cracking the Bluetooth PIN. In *Proceedings of 3rd International Conference on Mobile Systems, Applications, and Services (MOBISYS'05)*, pages 39–50, 2005.

[SWH98] P. Steggles, P. Webster, and A. Harter. The Implementation of a Distributed Framework to Support 'Follow Me' Applications. Technical Report TR.98.8, AT&T Laboratories Cambridge, 1998.

[TKS01] J. Trostle, I. Kosinovsky, and M.M. Swift. Implementation of Cross-realm Referral Handling in the MIT Kerberos Client. In *Proceedings of 8th Network and Distributed System Security Symposium (SNDSS'01)*, pages 109–124, 2001.

[TO03] S. Toner and D. O'Mahony. Self-organising Node Address Management in Ad hoc Networks. In *Proceedings of 8th International Conference on Personal Wireless Communications (PWC'03)*, pages 476–483, 2003.

[UPn03] UPnP Forum. Universal Plug and Play Device Architecture, v1.01 Draft. http://www.upnp.org/, 2003.

[W3C05] W3C Recommendation. XML Key Management Specification Version 2.0 (XKMS 2.0). http://www.w3.org/TR/2005/REC-xkms2-20050628/, 2005.

[WAP01] WAP Forum. Wireless Transport Layer Security Specification WAP-261-WTLS-20010406-a. http://www.wapforum.org/what/technical.htm, 2001.

[WB96] M. Weiser and J.S. Brown. Designing Calm Technology. *PowerGrid Journal*, 1(1):NA (online journal: http://powergrid.electiciti.com/1.01/), 1996.

[WCE05] Microsoft Windows CE. http://www.microsoft.com/windowsce/, 2005.

[WD01] A. Westerlund and J. Danielsson. Heimdal and Windows 2000 Kerberos - How to Get Them to Play Together. In *Proceedings of 2001 USENIX Annual Technical Conference*, pages 267–272, 2001.

[Wei91] M. Weiser. The Computer for the Twenty-first Century. *Scientific American*, 265(3):94–104, 1991.

[WFLY04] X. Wang, D. Feng, X. Lai, and H. Yu. Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint Archive, Report 2004/199, http://eprint.iacr.org/2004/146/, 2004.

[WHFG92] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1):91–102, 1992.

[WJH97] A. Ward, A. Jones, and A. Hopper. A New Location Technique for the Active Office. *IEEE Personal Communications*, 4(5):42–47, 1997.

[WS-02] WS-Trust Specification. Web Services Trust Language (WS-Trust). www.verisign.com/wss/WS-Trust.pdf, 2002.

[WS-04] WS-Policy Specification. Web Services Policy Framework (WS-Policy). ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf, 2004.

[WT99] A. Whitten and J.D. Tygar. Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of 8th USENIX Security Symposium*, pages 169–183, 1999.

[WYY05] X. Wang, Y.L. Yin, and H. Yu. Finding Collisions in the Full SHA-1. http://www.kriptopolis.org/docs/sha1.pdf, 2005.

[YBAG04] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password Memorability and Security: Empirical Results. *IEEE Security and Privacy*, 2(5):25–31, 2004.

[Ylo96] T. Ylonen. SSH - Secure Login Connections over the Internet. In *Proceedings of 6th USENIX Security Symposium*, pages 37–42, 1996.

[ZH99] L. Zhou and Z.J. Haas. Securing Ad hoc Networks. *IEEE Network Magazine*, 6(13):24–30, 1999.

[Zim95] P.R. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[ZMN03] F. Zhu, M. Mutka, and L. Ni. Facilitating Secure Ad hoc Service Discovery in Public Environments. In *Proceedings of 2003 IEEE Computer Software and Applications Conference (COMPSAC'03)*, pages 433–438, 2003.