The University of Kansas

Technical Report

A Framework for Sensor Networks with Multiple Owners

Satyasree Muralidharan, Dr. Victor S. Frost,
and Dr. Gary J. Minden

ITTC-FY2008-TR-41420-03

December 2007

Project Sponsor:
Oak Ridge National Laboratory

Abstract

A framework for sensor networks with multiple owners develops a mechanism for assured and controlled access to sensor assets owned and maintained by disparate organizations. The framework addresses the limitations in an existing system and proposes extensions to it. It also provides new mechanisms for cross-domain authentication and authorization by implementing a prototype as a proof of concept.

Table of Contents

List of Figures

1 Introduction

Latest developments in wireless communications attracted increased research upon sensor networks. The low-cost, low-power sensor devices often have limited computation and communication capabilities beyond the basic environment sensing. These capabilities coupled with growth in electronics opened up many technical issues in building networked systems based on them. This led to the design and architecture of systems like [1], [2], [3] and [4]. The objectives of [4], the SensorNet initiative were to: develop and/or discover the technology, standards, and technical requirements for an integrated national warning and alert system, to provide an incident discovery, awareness and response capability addressing local, regional, and national needs. SensorNet provides a standard mechanism to move information from sensors though the Internet to end user applications. Coordination of SensorNet activities has been lead by the Oak Ridge National Laboratory (ORNL).

Many efforts in the past addressed the design issues of various component technologies of sensor networks. PicoRadio [5], SmartDust [6] focused on system level issues in designing sensor hardware. LEACH [7] focused on network layer design issues for these networks. Other works like SensoNet [8] and WINS [9] recommend an entire protocol stack for sensor systems. Relatively few systems like SINA [8] discussed a model for sophisticated information dissemination systems that could be based upon the underlying sensor net technologies. At the SensorNet Architecture Forum [10], a unified architecture to enable the use of resources owned by disparate organizations to support the objective of SensorNet was discussed.

As sensor networks progress towards widespread deployment the security issues involved assume importance. Many early protocols like SNEP, µTesla [11] and Tesla [12] were proposed as building blocks to provide standard security functions to these networks. While work like [13] focuses on security solutions used for mobile user devices in the context of sensor networks, efforts like [14] considers a variety of approaches for key distribution in sensor networks by analyzing the overhead of these protocols on a variety of hardware platforms. Various research efforts were directed on providing a in low-end devices by integrating cryptographic primitives with low cost microcontrollers. For example: AVR controllers [15] and the Dallas iButton [16] support primitives for public key encryption, together with a possibility for modular exponentiation.

The above studies focused mainly upon the security functions that can be built inside a sensor node. They do not consider a broader security infrastructure for other components of sensor network architecture as the sensors are limited in resources to handle memory and computation intensive methods like asymmetric cryptography. Also they do not consider issues arising out of disparate ownership and cross policy domain resource access.

The research aim of this thesis is to develop one such framework that incorporates sophisticated authorization/authentication mechanisms that are secure and suited for disseminating and analyzing sensor information by extending a related system. The rest of the thesis is organized as follows. Chapter 2 provides an overview of the components and functionalities of the multi-owner architecture. Chapter 3 gives a background of an

existing system Ambient Computing Environment (ACE) [17], followed by Chapter 4 that discusses the extension of the existing system architecture to meet the new set of security and management requirements for a scalable and rapidly deployable sensor network. Chapter 5 presents a trust management with single and multiple organizations. Chapter 6 describes a prototype implementation of the proposed access control framework. Chapter 7 describes the conclusions and future extensions.

2 Multi-Owner Architecture

2.1 Overview

The objective here is to develop a unified architecture that has elements owned/controlled by a variety of organizations which can communicate across cross-administrative domains. The features of the architecture include:

- Assured and controlled access to sensor nodes in a multi-owner environment.
- Archiving and information dissemination.
- Application supporting high bandwidth requirements.
- Rapidly deployable sensor network.

This chapter gives an overview of the components and functionalities of the architecture.

2.2 Components and Functionalities

The architectural components are divided into three layers as shown in the figure 2.1 based on their functionality:

- Device Layer
- Repository Layer
- Application Layer

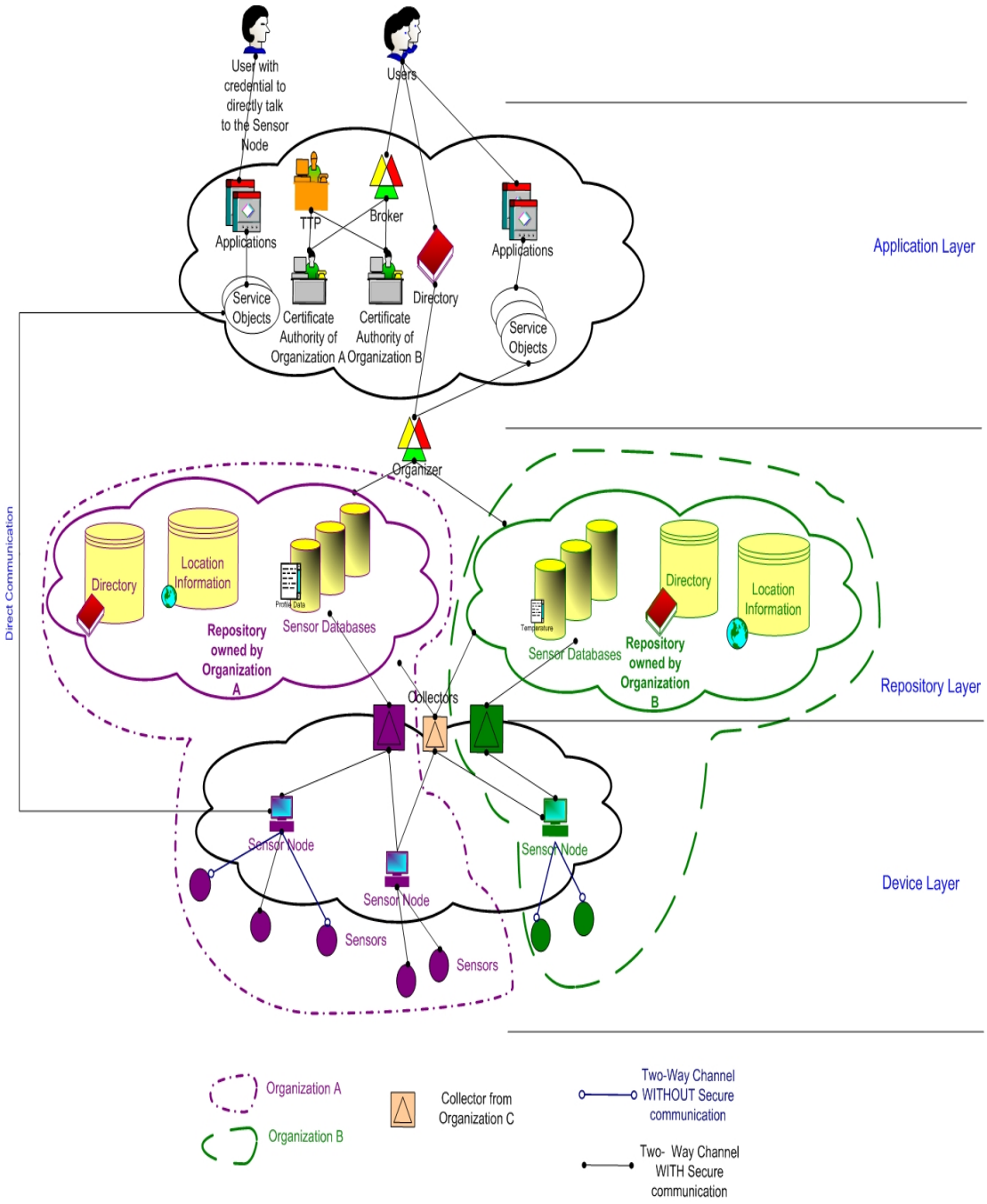

Figure 2.1: Multi-Owner architecture diagram representing the various components and their communications.

2.2.1 Device Layer

Device Layer composes of all the physical sensor endpoints together with the first level of data access and management points for the entire architecture. This consists of

- Sensors
- Sensor nodes
- Sensor services
- Collectors

Sensors are hardware devices capable of transducing a physical property e.g., temperature, pressure, light, humidity, etc., into an electric signal. Sensors communicate the collected data with the sensor node that controls them in the sensor network. The sensors could be of different types such as radiological, mechanical, optical or chemical sensors. Often sensors are characterized by small size and low energy consumption.

A sensor node is a computer that typically manages one or more sensors through a set of services. The sensors could be directly connected to the node either through serial or parallel ports or through a multi-hop network. The communication between the sensors and the node may or may not incorporate a secure communication. The security of this link depends upon the nature of the connectivity.

Sensor services are programs that control the sensors attached to the node. There could be one or more services per node, which each service dealing with one sensor. In the multi-owner architecture, there is a one-one mapping between the sensor service and the sensor controlled by it. In practical application, a single service could control more than one sensor. The architecture could be extended so that one service supports multiple sensors.

Collectors are programs that collect data from these services and transport them to the repository layer for further processing. There could be one or more collectors depending on the size of the device layer. The communication between the collectors and the sensor services follow the access control mechanism discussed later in this document. Collectors should authenticate and authorize themselves with the service, before tasking or configuring a sensor. Collectors collect data in one direction (from device to repository), other services load data or commands from repository to device, e.g., the Sensor Databases to the sensors to command the sensors. For example: loading a new smell profile in the Cyranose 320 Electronic Nose sensor (referred as Cyranose henceforth) from a Sensor database. Hence, Collectors could be viewed as intermediaries between the Device and the Repository layer. Collectors talk to the devices which typically belong to their organization or domain, our solution is not restricted for such a communication but spans across different organizational domains. Figure 2.1 shows sensors, sensor nodes and collectors from two organizations A and B.

2.2.2 Repository Layer

This layer stores the collected sensor data from the device layer. This composes essentially of databases which could be either:

- Sensor Databases that store and retrieve sensor data.
  Examples:
  - Database of the smell profiles, the response to a particular sample produced by the Cyranose.
  - Database of images captured by cameras used for surveillance.

- Infrastructural databases that store other information required to support the system. Examples:
    - Service Directory – database of current services available such as Temperature Sensor Service, Nose Service etc...
    - Regional Database – database of location of sensors.

There could be multiple repositories in this layer, each owned by a different organization. Figure 2.1 shows the repositories owned by two organizations A and B, where Organization A has a database of smell profiles along with the infrastructural databases and Organization B has a database of temperature data along with the infrastructural databases. The services from the device layer, register themselves with the Service Directory when they come online necessitating each organization to maintain a list of currently available services.

2.2.3 Application Layer

The application layer provides a unified view of the various components of the architecture. This layer consists of:

- Organizer,
- Applications,
- Service Objects,
- Users,
- Certificate Authorities,
- Broker and
- TTP

Organizer is a program that collects and transports data from the repository layer to application layer. A **user** in the multi-owner architecture is a human being who uses the infrastructure for various applications. A**pplications** are programs that can be either used to talk to the Organizer to get the processed data or to the Sensor Service directly as shown in the figure 2.1. All applications are written in Java and use Remote Method Invocation (RMI) [18] to communication with the sensor services. With JAVA-RMI, the services present themselves as remote objects. The applications get handles to these **remote objects** and use them to talk to the services.

A **Certificate Authority (CA)** is an entity that issues digital certificates. Each organization will have its own CA to issue certificates for users within that organization. The role of the CA is to issue certificates to identify the users and credentials to identify the actions that can be performed by the users. If a user wants to talk to devices from multiple organizations, then he/she needs to contact the CAs of different organizations individually to get credentials. As discussed later, we propose using a broker to avoid this requirement. A **Broker** is an entity that can issue credentials on behalf of CAs of different organizations. The broker is a delegated CA for the CAs of different organizations. A **Trusted Third Party (TTP)** is an entity who issues public key certificates for the CAs of organizations that trust it. The TTP help establish a chain of trust to authenticate users across multiple organizations. Chapter 5 describes the identification of users, credential distribution and delegation of trust when more than one organization is involved.

This 3-tier architecture is layered with organized communication between the layers using the intermediaries such as Collectors and Organizers. However, we anticipate that some scenarios might require a user talking directly to a device without having to pass through this layered architecture.

Direct Communication: Consider a situation where the user takes control over the sensors of all organizations and may wish to control them without having to talk to the organizer or the collector. In such a case, the user will get a single certificate to talk to devices from all organizations. The user will use the applications to talk to the sensor services controlling the devices through an out-of-band communication. Our solution also provides a way to have this **Direct Communication** between the user and the devices as in figure 2.1.

2.3 Challenges

In the multi-owner architecture, there are many organizations with each organization owning a large number of devices and supporting a large number of actions. Some of the challenges that need to be addressed in such a scenario are:

- How to provide control on the set of devices a client may use, even though the number of devices may be large?
- How to provide control over the set of actions a client can perform on devices he can access, even though the number and/or types of devices are large?
- How to provide clients access to devices controlled by multiple organizations?

2.4 Proposed Solution

In order to address the questions posted in the above section, we expand the Access Control framework provided by ACE [17], a system previously implemented at ITTC. ACE provides architecture for access control, but poses certain constraints for use in multi-owner environment as discussed in Section 4.1. We extend the access control mechanism used in ACE to achieve the granularity required for sensor networks with multiple owners.

This chapter provided an overview of the multi-owner architecture, the challenges and the proposed solution of extending an existing architecture ACE. The next section gives a background on ACE and highlights the features of ACE that are used for multi-owner architecture.

3 Background - ACE

Access to resources is usually achieved by programs running on specific processors. The ACE architecture was developed to untie the binding between programs and computers, and to create independent services so that the users can roam anywhere still preserving their sessions with the resources. ACE builds a pervasive system, where the users have long-lived workspaces and mobility within the environments irrespective of rooms or machines.

3.1 ACE Architecture

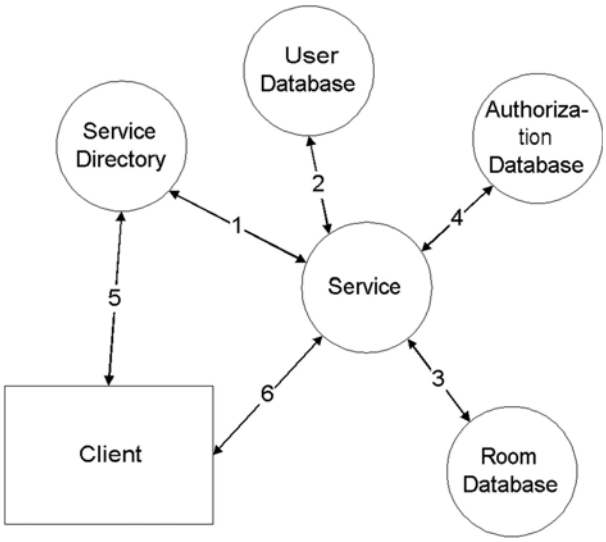

Figure 3.1: ACE Architecture showing the various components.

The services in the ACE architecture constitute the atomic level of computation. Services are grouped in to federations.
The communication between the service and the client are accomplished in two ways:

- Control channel: Provides a way for communicating control messages. It is a reliable in-order channel.
- Media channel: Provides a way for communicating audio and video. Reliability and in-order delivery are not important in this channel.

ACE allows the client to access resources through four core services as shown in the figure 3.1:

1. Service Directory: It is a directory service that locates all available services as well as their characteristics (Name, Location, and Service Class). All services register and un-register with this service. Since this is the directory for all the other services, the location of this service is fixed.
2. User Database: It is a database of all users in the system. The information includes Public Key, Name, Login name and Login characteristics. In ACE, the login characteristics include information like passwords, finger prints and iButton ids that can identify the user.
3. Room Database: It is a database of the information of all the rooms and building in the system.
4. Authentication Database: It contains the certificates of all the users in the system.

3.2 Key Features of ACE for the Multi-Owner Architecture

3.2.1 Client Server Communication using Enhanced RMI

Whenever a client wants to talk to the service, the client provides his credential showing that he has permissions to talk to the service to access the resources. The service validates his credential before providing the resource. The authorization is not only up to

the point of providing the resource, but also extends to every method that the client requests to perform on the device. In order to provide this per-method authorization, an enhanced form of RMI is used. The result of the enhanced RMI system is two classes:

- SecureUnicastServer.java: This handles new connections and creates a new client thread for each connection.
- ServerClientThread.java: This authenticates the user. When the user requests to perform an action, this thread does the per-method authorization, and then executes the requested method and returns the result.

The services present themselves as a Java remote object. The functionalities that the services advertise are given in the Java Interface. The client obtains the remote object to the service and performs actions using the Java RMI. This feature can be directly applied to multi-owner architecture, with the collectors being the clients, sensors being the devices and the sensor service being the gateway between the collectors and the sensors.

3.2.2 Secure Communication using TLS

Transport Layer Security (TLS) [19] provides authentication of the user and security of the message exchanges in the control channel. The users are identified by public key of the asymmetric key (either RSA [20] or DSA [21]). The keys are formatted to X.509 format [22], to be used by TLS. The public key is certified by a CA to verify the validity of the key. The user presents this signed certificate to any service for authentication. At the end of the handshake between the client and the server, a session key is negotiated and all the messages are encrypted with this key using any symmetric key algorithm such as Advanced Encryption Standard (AES) [23] or Data Encryption Standard (DES) [24].

In sensor networks with multiple owners, security of all the message exchanges between the elements of the architecture is important. This TLS and AES encryption of the ACE framework provides authentication of client-server, secured communication by encrypting the message exchanges and error-free delivery required in the communications between elements of the architecture.

3.2.3 Access Control using KeyNote Trust Management System

After user authentication, the exact permissions of the user determine the actions that the user can perform with the services. The service must authorize the actions requested by the user based on these permissions. ACE uses KeyNote Trust Management [25] to provide access control on the actions requested by the users. A very brief overview of KeyNote concepts and terminology that will be used in the remaining of the document follows.

3.2.3.1 KeyNote Terminology

KeyNote provides a simple language for describing and implementing security policies, trust relationships and digitally-signed credentials to control potentially dangerous actions over untrusted networks. Some of the Keynote concepts and terminology include:

- Assertions: Assertions describe the conditions under which a principal authorizes actions requested by other principal.
- Policy: Policy is one or more unsigned assertions.

- Credential: A credential is a signed assertion. A credential can be securely transmitted over untrusted networks.
- Action Attributes: Action Attributes are (name, value) pairs, and the primary objects on which KeyNote assertions operate. These names and values are arbitrary-length strings. KeyNote does not interpret the semantics of these names and values. These semantics must be agreed upon by the writers of applications and the authors of credentials.

```
KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Comment: Authorizer delegates read access to the Licensees
Conditions: ( app_domain ==”FileSystems” && file == “etc/passwd”) → “read”;
Signature: "sig-rsa-sha1-base64:XQZopw.."
```

Figure 3.2: Example of a credential in KeyNote.

Figure 3.2 shows an example of a credential. “Authorizer” identifies the principal authorizing actions to users identified in “Licensees”, under the “Conditions” that in the application demanding authorization, the action attributes set includes:

- The attribute named “app_domain” with value “FileSystems” and
- The attribute named “file” with value "etc/passwd".

The authorization level is “read”. KeyNote provides a Compliance Checker engine that evaluates credentials and returns the result (an application-defined string). The application can then decide what to do depending on the result given by the KeyNote engine. ACE uses KeyNote Trust Management to authorize the client actions with the device. In ACE, the administrator “ace” is given all rights to use the system. This “ace” administrator is provided a policy which is implicitly trusted and does not have to be signed.

```
KeyNote-version: 2
authorizer: POLICY
local-constants: KEY1 = "x509-base64:MIIEZzCCA...LCSG0N2ICh"
licensees: KEY1
conditions: (APP_DOMAIN == "ACE") -> _MAX_TRUST;
```

Figure 3.3: Example of a policy in ACE.

Figure 3.3 shows an example of a policy. Figure 3.4 shows an example of a credential used in the ACE environment.

```
KeyNote-version: 2
authorizer: "x509-base64:MIIEZzCCA9CgAw...LCSG0N2ICh"
licensees: KEY1 = "x509-base64:MIIEZnb53...ighfkRT4523k"
conditions: ((APP_DOMAIN == "ACE") &&
(time >= 1082390980610) && (time <= 1082390980628)) -> "write";
signature: "sig-rsa-sha1-base64:Nt4+XIP...soP+mgjjTXWA=="
```

Figure 3.4: Example of a credential in ACE.

A CA is used to issue public key certificates and credentials. In ACE, the “ace” administrator acts as the CA. All other users get a credential with the “ace” administrator being the authorizer.

This trust management allows the multi-owner architecture to control access to the actions performed by a collector on the sensors. Each method that the client is trying to access through the remote object of the service can be checked for authorization by querying the KeyNote engine. If the collector does not have a valid credential, he cannot perform the requested action on the sensor and an exception is raised. Since this authorization is implemented within the service infrastructure, all services inherently implement the authorization procedure. The examples of policy and credential show the expressiveness and ease of representation of security policies and credentials using a unified representation language.

Although it is possible to conceive an equivalent system with popular key distribution and trust management systems like Kerberos, KeyNote offers significant advantages. The following section highlights and discusses these advantages in more detail.

3.2.4 Choice of KeyNote Trust Management

Kerberos was initially designed for symmetric key distribution and authentication. It included a delegation and authorization mechanisms which are not as sophisticated as KeyNote. The following section introduces Kerberos followed by a comparative discussion between Kerberos and KeyNote.

Introduction to Kerberos:

Kerberos is a secret key based service for Authentication designed by MIT [26]. Kerberos achieves authentication of a user to access remote resources. The entities in Kerberos are:

- The Key Distribution Center (KDC),
- Principal or user and
- An Application to authenticate users.

The KDC shares a master key (symmetric cryptography) with each of the principal. Kerberos achieves “Decentralization” by dividing the network into realms. A realm is a collection of resources, users and a single KDC to manage. Each realm has its own KDC. Some of the deficiencies in Kerberos include:

Cross-Domain Authentication:

In the sensor network architecture with sensors owned by multiple organizations, cross-domain authentication is important. Cross-realm authentication in Kerberos is expensive in terms of administrative effort. All the users in each realm should know the trust relationships between realms in order to find a path of KDCs in between, to reach the intended destination. Referrals [27] can be used to interactively determine which KDCs to contact for tickets to establish a trust path to the destination. This comes with an assumption of using the domain name of the host for identifying the next KDC in the trust path. In KeyNote there is regular hierarchy of trust. Users from different domains can easily find a common CA up higher in the hierarchy to establish a trust relationship.

This feature will be essential to reduce administrative efforts in establishing trust paths in the multi-owner architecture.

Delegation of Tasks:

Delegation is another important aspect especially when there is a large network of sensors and cross-domain communications. In Kerberos, handling vertical separation of duties is difficult as there is no organized hierarchy of realms. If a user A has to delegate his task to user B, he needs to contact KDC to get a TGT (Ticket Granting Ticket - This includes {Identity, Session Key, Expiration time} signed by the KDC's master key), give this TGT to B, which is used by B to contact the KDC again to issue tickets. For every delegation event, the KDC is contacted twice which is a disadvantage for performance reasons. The extra interaction with KDC allows it to know about a delegation event and enables the KDC to audit delegation events [28]. With KeyNote, delegation is simple with hierarchical arrangement of Certificate Authorities. Any user can delegate authority to anyone else in the network by issuing a certificate (also known as credential signed by this user) without having to contact any Certificate Authorities at all. The event of delegation is known only when the delegated user uses his certificate with an application to perform some action.

Authorization:

Kerberos was initially designed for authentication only. It authenticates the user, but the actions that the user performs on the resources are not validated. Kerberos was improved to support authorization by providing a field "AUTHORIZATION-DATA" in the TGT or tickets. For cross-domain delegation, this poses a difficulty in access-control as this authorization information should be included in each ticket issued by the intermediate KDCs and the destination has to contact each of the intermediate KDCs in the path for authorizing the user. This requires the KDCs to be online at all times, and also slows down the authorization procedure. KeyNote was designed mainly for authorization of actions performed by users. When the user requests an application to perform some action, the application submits the users request along with his certificate to the KeyNote Engine, which performs a compliance-checking and gives the result back to the application. The application can then decide whether to allow the user to continue or not. It helps the application in this decision-making procedure. For cross-domain authorization, in order to verify the certificate of the delegated user, KeyNote requires:

- The public key of the CA who signed this certificate,
- The certificate of this CA,
- The public key and the certificate of the CA who issued certificate to the above CA. This is repeated until the root CA's certificate is reached.

Since all the public key certificates are available online, the authorization procedure is fast compared to Kerberos. Speeding up the authorization procedure is essential for the multi-owner architecture, because the communication between a Collector and the Sensor Service is time critical.

Synchronization of Clocks:

Another disadvantage of Kerberos is that the nodes in the network should have their clocks reasonably synchronized (within five minutes) since the users authenticate each

other by verifying the timestamp encrypted with the shared session key issued by the KDC. There is no need of clock synchronization in KeyNote. It is expected that the nodes in sensor networks will be loosely synchronized further motivating the use of KeyNote.

In addition to the above advantages of KeyNote, it has a simple mechanism for message exchanges when implementing authorization. The following section describes the protocol of the message exchanges used in ACE.

3.3 Access Protocol in ACE

This section describes the sequence of actions when an ACE user talks to an ACE service. Each time a new client contacts a service, the service spawns a new thread dedicated to the communication with the specific client.

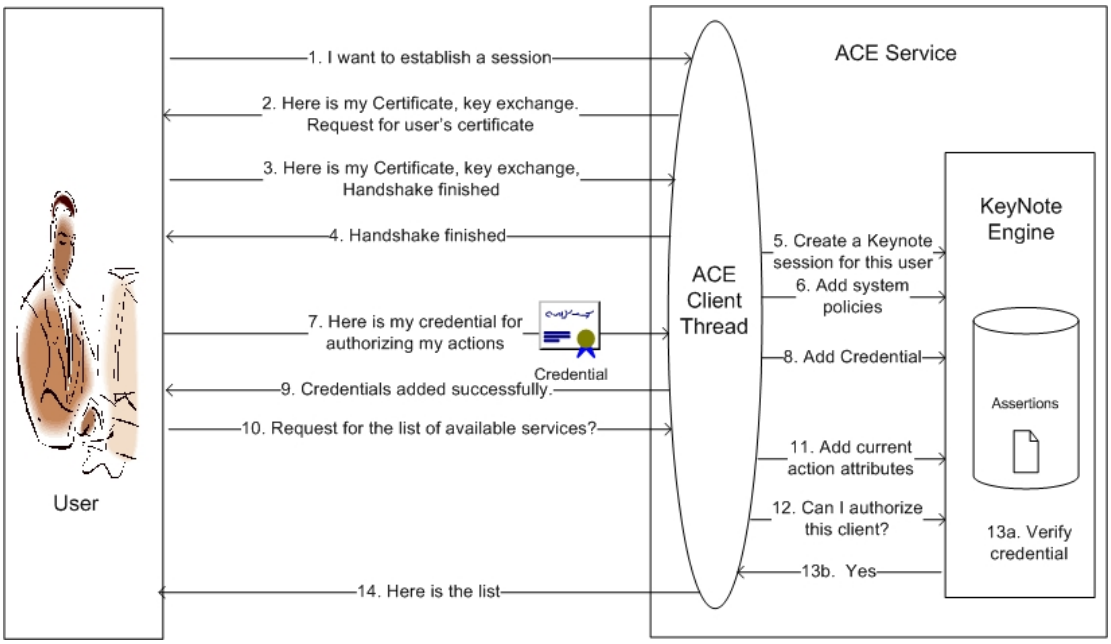

Figure 3.5: Access Protocol showing the sequence of actions when an ACE Client contacts the ACE Service.

Typically the following happens when a client wishes to access a resource:

1. User contacts the service (the client thread of the service) for establishing a session.
2. The service replies with its certificate for authentication, key exchange for establishing session key and request the user for his certificate.
3. The user replies with his certificate and session key exchange. The user verifies the Server's certificate. The client sends "Finished".
4. The service sends "Finished". The TLS authentication is completed and a session is established.
5. The service creates a new KeyNote Session that will be used for user's authorization.
6. The service provides the required policy to KeyNote database to be used to verify the client's credentials later.
7. The client provides his credential to the service.

8. The service adds this credential to the KeyNote database.
9. The service replies the result of adding the credentials to the user.
10. The client requests the service for an action. Example: The client requests the service to list the available services.
11. The service provides the KeyNote, the current set of action attributes such as domain in which this application is used, room in which the service runs, current Time and the method requested by the client.
12. The Service queries the KeyNote for authorizing the action requested by the client.
13. The KeyNote engine verifies the credential against its current set of attributes and returns the result to the Service.
14. The Service determines whether to perform the action requested by the client or not depending on the result from the KeyNote engine. If the Service performs the action, it returns the result to the client. Else, it returns an "Access Denied" to the client.

Figure 3.5 shows the sequence of actions when an ACE client contacts the Directory Service.

This chapter described the security features of ACE namely enhanced RMI for communication, TLS authentication and KeyNote authorization that are required for multi-owner environment including a discussion on choice of KeyNote over Kerberos. The RMI, communication and TLS authentication are used as such for the communication of the sensor elements in the multi-owner architecture, but extensions are made to ACE authorization mechanism to meet the security requirements of multi-owner environment and these extensions are discussed in the next chapter.

4 Extending ACE for Access Control for Multi-Owner Architecture

KeyNote authorization is based on the action attributes in the credentials. A user is authorized if the conditions in the credential match the current set of action attributes that the service provides the KeyNote engine. It would be a significant challenge to define a set of attribute names and the meaning and syntax associated with the values for each one that would effectively address the requirements for an ever growing number of organizations and sensor devices. Instead we propose a core action attribute set for the multi-owner architecture by extending the action attribute set provided by ACE. This chapter describes the limitations of ACE and the extension made to the ACE set of action attributes for the multi-owner architecture.

4.1 Limitations in the ACE Framework

The integration of KeyNote into the ACE framework ensures that each client invocation of a service method is authorized. This provides a firm foundation and a base for access control in the Sensor Network environment. There is no possibility to invoke a service method without providing a KeyNote credential for allowing authorization. However, the action attribute set utilized by services in the ACE framework is directly embedded at a low level in the program infrastructure. This set of action attributes is

limited, and does not allow the flexibility to conveniently define credentials that will provide controlled access to a wide variety of sensors in the multi-owner environment.

4.1.1 Current Set of Action Attributes in ACE

The set of attributes provided by ACE for access control include:

a. APP_DOMAIN – Domain for which these credentials are used.
b. Time – Time of request
c. Method – Method requested to be performed on the device
d. Room – Room in which the device is found

Sensor Network involves distribution of credentials with different levels of specificity, i.e., it involves situations requiring a single credential for a group of devices and situations involving credential for a particular action of a device. ACE cannot define a credential that allows client to execute some method, xxx, on only one of two devices in the same room (because we cannot distinguish between devices of the same type in the same room). It cannot provide a user with "read" access to all devices in a room, it would be necessary to identify all the methods for all types of devices in the room and include them all in the credential thus making the credential large.

4.2 Extension to the ACE Set of Action Attributes

The above set of action attributes have to be extended to provide a broader set of action attributes to support more expressive conditions in KeyNote credentials. This section describes the names, syntax and meaning of the core action attribute set for the multi-owner architecture along with a mechanism for extending this set.

4.2.1 Core Action Attributes for Multi-Owner Architecture

The new set of action attributes include:

1. **APP_DOMAIN** – Domain for which these credentials are used. This attribute is retained from the current set.
2. **ServiceClassHierarchy -** Full Java class hierarchy path for the service performing the action.
3. **MachineName -** The name of the machine in which the service is running.
4. **Method -** Simple (not full) Java method name the client is attempting to execute. This attribute is retained from the current set.
5. **FirstArgValue -** Specialized attribute assigned the value of the first argument supplied to the method to be executed, provided that the method has at least one argument; and the first argument is an instance of class String, otherwise it is empty
6. **Role -** Attribute identifies a general level of device control that the client must be authorized to perform an action. It is assigned a value from the following set of alternatives:
   - Reader – one who can retrieve information from the device
   - Writer – one who can load information on to the device
   - Administrator – one who can perform (almost) any action on the device
   - No role attribute is acknowledged for the specialized action.

The attribute Role, is used to identify meaningful groupings of service methods. The values associated with this attribute is widely used and well understood.

Each action (method) performed that the Sensor service provides, is associated with the Role requirement:

1. Reader is allowed to view information from the service/device, for example, getSensorReading().
2. Writer is allowed to execute methods which modify the state of the device/service and are intended for usage by a range of clients, for example, setSensorProfile(newProfile).
3. Administrator is allowed to perform almost all methods provided by the service. This role implicitly provides access to all methods permitted to either "Reader" or "Writer" as well as methods that are reserved for a client with a higher level of authorization, for example, resetSensor(). The credential will state what role the holder of the credential could perform on the device. The role could be combinations of "OR"s when the holder could do multiple roles on the device. If the client is an "Administrator" the credential will say (Role == "Reader" || Role == "Writer" || Role ==”Administrator") as the "Administrator" role supersets the role of "Reader" and "Writer".
4. Methods in the system might have either "Reader", "Writer", "Administrator" roles or no role associated with them. Methods that have a role associated could only be accessed by the user carrying that particular role in their credential. For example: a user with "Reader" role will be able to access a method that has an associated "Reader" role attribute. It should be noted here that the user need not have that method name mentioned in his credential. However, for accessing those methods which DO NOT have a role attribute associated with them can ONLY be accessed by explicitly mentioning the method name in a user's credential. Figure 4.1 shows a credential of user whose role is an "Administrator".

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
((app_domain == "SensorNet") && (Provider == "ITTC") &&
(Time <= "1151465644580" && Time >= "1161465647580") &&
(ServiceID == "ChemicalSensor001") && (Role == "Administrator" ||
Role == "Reader" || Role == "Writer")) → "allow";
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.1: A credential where the user has a role "Administrator" on a Chemical Sensor.

It is also to be carefully noted that this "Administrator" role is different from the "Super Administrator" in the system that has a system-wide access. Figure 4.2 shows the assertion for a Super Administrator.

```
KeyNote-version: 2
authorizer: POLICY
local-constants: KEY1 = "x509-base64:MIIEZzCCA...LCSG0N2ICh"
licensees: KEY1
conditions: (APP_DOMAIN == "SensorNet") -> _MAX_TRUST;
```

Figure 4.2: A policy of a Super Administrator.

It is now clear that although a user might have the role of "Administrator", he will not be able to access every method in the system. The method shutdown() on all the sensors could be an example of such a priviledged method that needs special mention in the credential. Considering an emergency scenario, a user can get access to perform ONLY THIS priviledged method on all the sensors of all the Organizations. He can be given a credential in which case the credential should be signed by a trusted third party trusted by all the Organizations.

The credential would need to include a condition: (ServiceClassHierarchy == "<full-hierarchy-spec>" && Method == "shutdown").

Each Organization needs to decide the set of such privileged methods that needs special mention in the credentials.

7. **ServiceID** – A unique identification for each instance of a service.
8. **Time -** Time of request. This attribute is retained from the current set.

Of the above set of attributes, attributes that identify the actions that the client can perform include: ServiceClassHierarchy, MachineName, Method, FirstArgValue and Role. Attribute that identify the set of devices or the services that the client can talk is ServiceID.

Based on their nature, the above set of action attributes can also be classified into two categories:

1. External Attributes
2. Internal Attributes

4.2.1.1 External Attributes

Attributes that are placed outside the service program are external to the service. The majority of the attributes listed in this document are **static** (do not change for a service or device) for the duration of the service session, in particular all those that relate to identifying the device/service. Therefore, these (name, value) pairs can be stored in a file and read when the service starts. This approach provides much more flexibility of the owner of a device to control the syntax for attribute values (such as DeviceName) and even the number of attribute (name, value) pairs. Providing additional elements to the action attribute set will never cause a previously valid credential to fail. Since this file has a direct effect on which credentials will be valid (or fail) for the associated service, it is critical that access to this file be carefully controlled. Few examples of external static attributes are "ServiceID" that does not change for a service, "APP_DOMAIN" that does not change for an Organization. There are **no external dynamic** attributes as attributes that are dynamic can be only obtained programmatically and thus cannot be outside a service.

4.2.1.2 Internal Attributes

Attributes that can be obtained programmatically by the service are internal to the service. Internal attributes can be either static or dynamic.

Static Internal Attributes:

Attributes that do not change for a service such as the service class hierarchy, Java package name and machine name of the computer in which the service runs are static for a service, but changes from one service to another. For example: "ServiceClassHierarchy" and "MachineName" that are obtained in the program once the service is started and registered with the Service Directory. Since these attributes are obtained programmatically, this set of action attributes and their value syntax are more difficult to change or extend (requires program modification).

Dynamic Internal Attributes:

Most of the elements of the action attribute set are static. Those that are related to identifying the specific method the client requests to execute such as, "Method", "FirstArgValue" and "Role" associated with each method are not, however. Likewise, the attribute "Time" changes with each request. These elements of the action attribute set must also be assigned programmatically. Therefore, the dynamic elements of the action attribute list and their value syntax are also significantly more difficult to change or extend (requires program modification). Testing and evaluation has shown that there will be little need to modify this set.

4.2.2 Examples

A few examples of the credential with the new set of action attributes showing their granularity and their expressiveness are given below.

Example - 1:

Consider a case where there is a user from Organization "A" who owns one particular sensor. This user is the administrator of this sensor. With cross-domain communication, he could be given access to read sensor data from all sensors of a particular type owned by a different Organization "B" in the network. Figure 4.3 shows the credential that could be given in such a case. The user has an "Administrator" role on one particular Chemical Sensor owned by his Organization "ITTC" and a "Reader" role on all cameras owned by Organization "EECS".

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Comment: The user of this credential has a "Reader" role on all cameras of Organization "EECS" and "Administrator" role on one particular Chemical Sensor of Organization "ITTC".
Conditions:
((app_domain == "SensorNet") && (Provider == "EECS") &&
(Time <= "1151465644580" && Time >= "1161465647580") &&
(ServiceClassHierarchy ~= "^.*Camera$") && (Role == "Reader")) → "allow";
((app_domain == "SensorNet") && (Provider == "ITTC") &&

(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == “ChemicalSensor001”) && (Role == “Administrator” || Role == “Reader” || Role ==”Writer”)) → “allow”;
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.3: A credential where the Authorizer delegates the Licensees “Reader” role on all cameras and “Administrator” role on one particular Chemical Sensor.

The use of regular expressions in KeyNote credentials avoids listing all the cameras of Organization “EECS”, and the attribute “Role” clearly defines what the user can do with each of these organizations.

For the first clause, the ActionAttributeSet file will contain:
Provider = “EECS”
ServiceID = “VCC3Camera009”
And the internal attributes that should result in an “allow” will be:
Time = “1151465647580”
Role = “Reader”
Method = “getCameraTiltAngle**”**
ServiceClassHierarchy = “SecureUnicastSever.Base.Service.
Device.PTZCamera.VCC3Camera”
MachineName = “barney.ittc.ku.edu”
For the second clause, the ActionAttributeSet file will contain:
Provider = “ITTC”
ServiceID = “ChemicalSensor001”
And the internal attributes that should result in an “allow” will be:
Time = “1151465647590”,
Role = “Administrator”
Method = “resetSensor**”**
ServiceClassHierarchy = “SecureUnicastSever.Base.Service,
Device.ChemicalSensor.Nose”
MachineName = “terbium.ittc.ku.edu”

Example - 2:

Consider a case where an Organizer collects data from databases of Organization A and Organization B and maintains a database of his own. Figure 4.4 shows the credential that can be used in such a case. The user can read sensor data from a Sensor Database of Organization “ITTC” and modify a Service Directory database of Organization “EECS”.

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
((app_domain == “SensorNet”) && (Provider == “EECS”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == “ServiceDirectory001”) && (Role == “Writer”)) → “allow”;

((app_domain == “SensorNet”) && (Provider == “ITTC”) &&

(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == “SensorDatabase002”) && (Role == “Reader”)) → “allow”;
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.4: A credential where the user can modify Service Directory database and can read sensor data from a Sensor Database.

The credential is simple and captured all the attributes required to identify the Organizer’s action with both the organizations.

Example - 3:

Consider a case of a typical action of a Collector collecting data from a sensor and updating a Sensor Database belonging to his organization. Figure 4.5 shows the credential that can be used in such a case. The collector can read sensor data from a Radiological Sensor and update a Sensor Database of Organization “ITTC”.

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
((app_domain == “SensorNet”) && (Provider == “ITTC”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
((ServiceID == “RadiologicalSensor001”) && (Role == “Reader”)) ||
((ServiceID == “SensorDatabase002”) && (Role == “Writer”))) → “allow”;
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.5: A credential where a collector can read sensor data from a Radiological sensor and update a Sensor Database.

The credential is simple and any user holding this credential will be able to identify for what this credential was intended by reading through it.

Example - 4:

Consider a case where a Collector configures a sensor and collects data from the sensor belonging to his organization. Figure 4.6 shows the credential that can be used for this case. The Collector has the role “Reader” or “Writer” access on the temperature sensor in a computer “sentinel.ittc.ku.edu”.

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
((app_domain == “SensorNet”) && (Provider == “ITTC”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == “TemperatureSensor350”) && (MachineName == “sentinel.ittc.ku.edu”)
&& (Role == “Reader” || Role == “Writer”)) → “allow”;
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.6: A credential of user with “Reader”/“Writer” role on a Temperature sensor.

Listing all the methods that the temperature sensor provides for configuring and methods to read sensor data from the sensor would result in a large credential. With the use of "Role" attribute, the credential captured the requirement of the Collector in a single line. The attribute "MachineName" identifies the computer on which the sensor service is available.

The corresponding ActionAttributeSet file will contain:

Provider = "ITTC" and ServiceID = "TemperatureSensor350"

And the internal attributes that should result in an "allow" will be:

Time = "1151465647580"
Role = "Reader"
ServiceClassHierarchy=
"SecureUnicastServer.Base.Service.Device.Sensor.TemperatureSensor"
Method = "getCurrentTemperature"
MachineName = "sentinel.ittc.ku.edu"

If the collector requests for method "setScale("Celsius")", then the internal attributes that should result in an "allow" will be:

Time = "1151465647580"
Role = "Writer"
ServiceClassHierarchy=
"SecureUnicastSever.Base.Service.Device.Sensor.TemperatureSensor"
Method = "setScale" and MachineName = "sentinel.ittc.ku.edu"

Example - 5:

Consider a case where all sensor services of an organization follow a standard interface. They provide a common method that can be used across all the sensor services. The parameter of the method decides the action the user can perform on the service. In such a case, the attribute FirstArgValue comes in handy.

KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions: ((app_domain == "SensorNet") && (Provider == "ITTC") && (Time <= "1151465644580" && Time >= "1161465647580") && (ServiceID == "ChemicalSensor350") && (Method == "execute") && (FirstArgValue == "FetchResults")) → "allow";
Signature: "sig-rsa-sha1-base64:XQZopw.."

Figure 4.7: A credential explaining the usage of FirstArgValue.

Figure 4.7 shows an example of a credential with the FirstArgValue. The user holding this credential can load a profile to the chemical sensor provided by "ITTC". All the sensor services in "ITTC" follow a standard interface and provide a method called "execute". The value provided to the first argument of this method identifies the action that the user can perform on the sensor. The actions could be either loading or retrieving the sensor data. In this case, the user is retrieving the results of the last identification

represented by the FirstArgValue. Thus, the access to the sensor service could be restricted even to the parameter of the method executed by the client.

Example -6:

Consider a case of an emergency where the user is given permissions to shut down all the sensors of all organizations. Figure 4.8 shows the credential that can be used in such a case. All the organizations have a trust relationship with the authorizer of this credential. The user has rights to perform the privileged method "shutdown" on all the sensors of Organization "EECS" and "ITTC".

```
KeyNote-Version: 2
Authorizer: "x509-base64:MIIA.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
(( app_domain == "SensorNet") && ( Provider == "ITTC" || Provider == "EECS" ) &&
(Time <= "1151465644580" && Time >= "1161465647580") &&
(ServiceClassHierarchy~= "^.*Sensor$")") && (Method == "shutdown")) → "allow";
Signature: "sig-rsa-sha1-base64:XQZopw.."
```

Figure 4.8: A credential where the user can perform only one method on all the sensors.

The examples provided in this section described the use of the new set of action attributes proposed for the multi-owner architecture. It was shown that this new set of action attributes helps expressing the credential in a simple and meaningful way.

This chapter described the extensions that were made to ACE set of action attributes to meet the granularity required for access control within the context of a single organization. However, additional challenges arise when more than one organization is involved with regards to credential distribution and delegation of trust. The next chapter discusses these issues.

5 Access Control in Multi-Owner Environment

The access control sub system of the architecture follows a "Take-grant Protection Model" [29], where in the user presents his capability to request an action. This is in contrast to "Rule Set Based Access Control" model [30], where in the system trust the user's discretion in performing an action. While the latter is desired and sufficient for a tightly coupled system (e.g., Linux Kernel), the former is more useful for a loosely coupled distributed system like sensor networks. The core of the access control subsystem is the ability to issue the authorization information the form of a "credential". However, when we analyze the presentation of credentials during service access, and the granularity of controlled objects, the following questions are exposed:

1. How to verify a user identity?
2. Who issues credentials across organizations?
3. How does the access control mechanism work when a user talks to a sensor belonging to his organization?
4. How does the access control mechanism work when a user talks to a sensor belonging to a different organization?

It is known that in public key cryptography, the users have public keys. Public keys are accessible by anyone in the network. CA is used in combination with certificates and protocols to provide authentication and authorization functions. We assume that each organization maintains its own CA. The CA issues certificates which are signed messages that map the user and his public key. CA is the public key equivalent of KDC. Broadly the functions of CA include:

1. Signing the public keys of the users
2. Creating credentials for users to talk to devices

5.1 Scenario with CA from a Single Organization

Considering the access control mechanism in a single organization, there is one CA that issues certificates and credentials to users in this organization. The role of CA ends with issuing the credential and the certificates. The CA need not be contacted when a user authenticates and establishes session with the sensor service assuming that the service trusts the public key of the CA. It is enough if the authenticating sensor service knows and trusts the public key certificate of the CA for signature verification. The issued certificates and credentials remain with the user. When a user requests for a credential to access devices, the CA must be aware of the list of sensor services and functionalities of each of the service that the organization provides. If the CA does not know the specifics of a service, then it cannot generate a credential that provides fine-grained access control. The extensions to KeyNote credentials discussed in the previous chapter poses a constraint on the CA. Each service has a set of functions and the role that is required for perform that action. Every time there is a change in the service, the Credential Author (or CA) has to talk to the Service Author to get the information on the new capabilities and their corresponding roles. In the current prototype, this interaction between the CA and the Service Author occurs manually. This interaction can be automated by developing a utility that helps the Credential Author to look up the service and list the current set of methods and their roles.

5.1.1 Revocation of Public Key Certificates and Credentials

All public key certificates have an issue date and expiration date. This interval is determined by the organization. The typical validity interval is about a year, to avoid the nuisance of renewing certificates with smaller validity periods. The revocation becomes more pronounced in the event of users entering and leaving the organization, for example: a user getting fired while he still holds a certificate that is not yet expired.

A solution to certificate revocation could be to use Certificate Revocation Lists (CRL). A CRL is a list of serial numbers that should not be honored. A certificate is valid only if:

- It has a valid CA signature
- Not yet expired
- Not listed in the recent CRL

The authenticating service should check the recent CRL as a part of the authentication handshake. The use of X.509 certificates for TLS authentication provides an easier way to tackle this problem. Any X.509 public key certificate has a serial number along with the version, signature algorithm identifier, validity period, issuer, subject, public key of subject. The CA that creates the certificate is responsible for

assigning it a serial number to distinguish it from other certificates it issues. When a certificate is revoked, its serial number is placed in a Certificate Revocation List (CRL). The prototype implemented here does not provide a solution for the above problem, but considers the pros and cons of using Certificate Revocation Lists as a part of the future research tasks.

With KeyNote, credentials are monotonic. Removal of an assertion does not cause an increase or decrease in the compliance value. KeyNote strictly does not support negative credential (credential that can invalidate the credential already given). Hence, once a credential is issued, it is valid until the credential expires, and cannot be revoked. This might create a problem when there are misbehaving users in the system resulting in insecure access control. The presence of misbehaving users determines the significance of revocation.

5.2 Scenario with CAs from Multiple Organizations

Considering a scenario where there are multiple organizations A, B and C. Each organization has its own CA to issue certificates and credentials to its users. In a sensor network with multiple owners, it is likely to see that a user from one organization would be talking to devices owned by other organizations. Considering a case where a user from an organization A wants to talk to devices from Organization B and C. He cannot use his public key certificate signed by his own CA with a service from a different organization, if there is no trust relationships between the organizations. A straight forward solution to such a case will be that the user to contact each of the Certificate authorities individually.

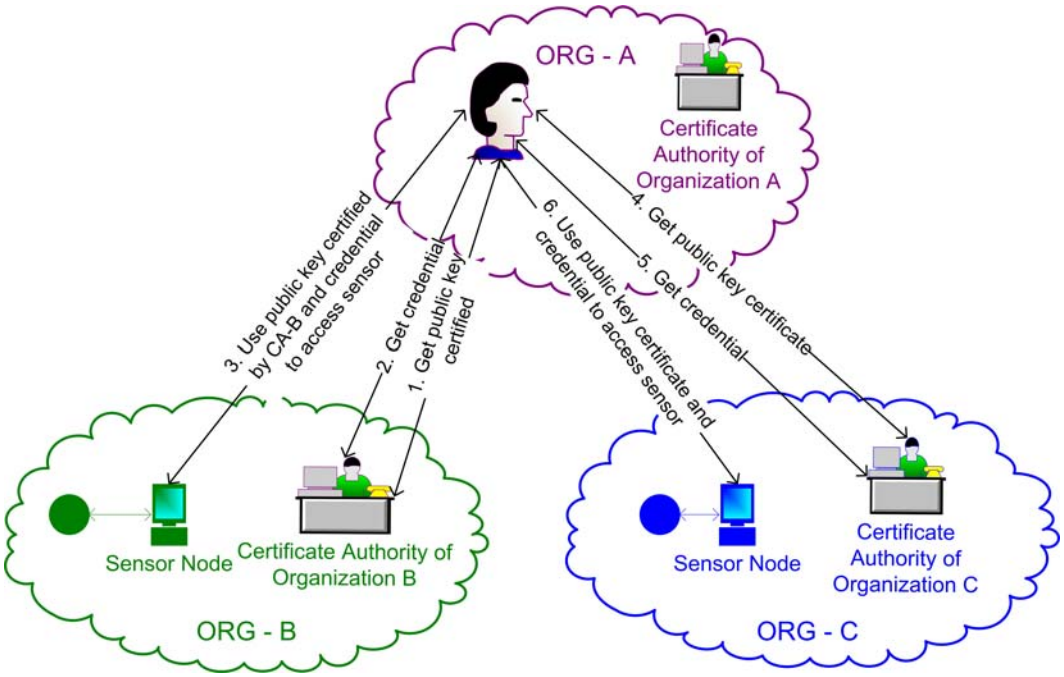

Figure 5.1: A scenario showing the complexity when a user contacts CA of different organizations individually for credentials.

The user from Organization A contacts the CA of organization B. He gets his public key certified by this CA. Then he places his request for accessing the sensor of organization B and gets a credential. He uses this credential to talk to the sensor service of Organization B. This transforms into scenario of a single CA within an organization, as the user holds a separate public key certificate and credential for Organization B. He repeats the above steps with Organization C to talk to sensor service of Organization C. The current prototype implementation has a framework to demonstrate this solution. But there are serious disadvantages of using such a straight forward solution:

- When the user wants to talk to multiple devices from multiple organizations, then he has to get his public key certified by CA of each organization. The user has to maintain a large set of his public key files signed by each of the CA. Users will have difficulty in managing huge number of such files.
- Similarly, the user has to get one credential for each of the organizations. This is highly not scalable and inefficient as the user has to manage and know what file to present to a service. Figure 5.1 describes the complexity of contacting multiple CA individually for credentials.
- The users must know the presence of all the different CA in the network.
- Moreover, the CA of a different organization may not trust this user and may not sign his public key.

5.3 Proposed Solution

The above scenario presents the difficulties of authentication and authorization during a cross organization service access. We can address these by introducing two entities: a Trusted Third Party (TTP) and a Broker. The primary function of a TTP in the architecture is to enable users to authenticate across various domains, whereas the function of a Broker is to provide a user with authorization for a requested action on behalf of participating organizations. The roles are complementary and distinct. In reality both of these roles can be assumed by any user of the system or an outside entity who is able and trusted. The following sections describe the role and use of TTP and Broker in more detail.

5.3.1 Authentication using TTP

In the absence of a trust relationship between users and services belonging to different organizations, we need a TTP to establish such a relationship for authentication purposes. This section outlines the steps involved in achieving such an authentication scenario.

5.3.1.1 Role of a TTP

A TTP facilitates authentication between organizations that trust it. They use this trust to identify the users among organizations. The role of a TTP is issuing public key certificates to CAs of the trusting organizations. A CA, in turn issues public key certificates to users within its organization.

5.3.1.2 Limitations in the Current Authentication Handshake

The authentication in ACE is achieved using a CA for each organization. The CA of each organization has a self-signed certificate. A self-signed certificate is one for which

the issuer (signer) is the same as the subject (the entity whose public key is being certified). Users within the same organization are certified by the CA, and hence are trusted by the services within the organization. A user from another organization cannot authenticate to these services, as the services do not have any knowledge about the CA of that organization. Hence, the current authentication setup is limited to users within the same organization and cannot accomplish a cross-domain communication. The establishment of certificates used in the current setup is extended using “Chain of Trust” provided by the Public Key Infrastructure (PKI) to achieve the cross-domain authentication which is discussed in the next section.

5.3.1.3 Authentication using a Chain of Trust

Consider a scenario where two Organizations A and B requires cross-domain authentication. The TTP is trusted by both of these organizations. The TTP has a self-signed certificate. The TTP signs the public keys of the CAs of both the organization. The certificate issued by the TTP and the self-signed certificate of the TTP forms the certificate chain identifying the CA.

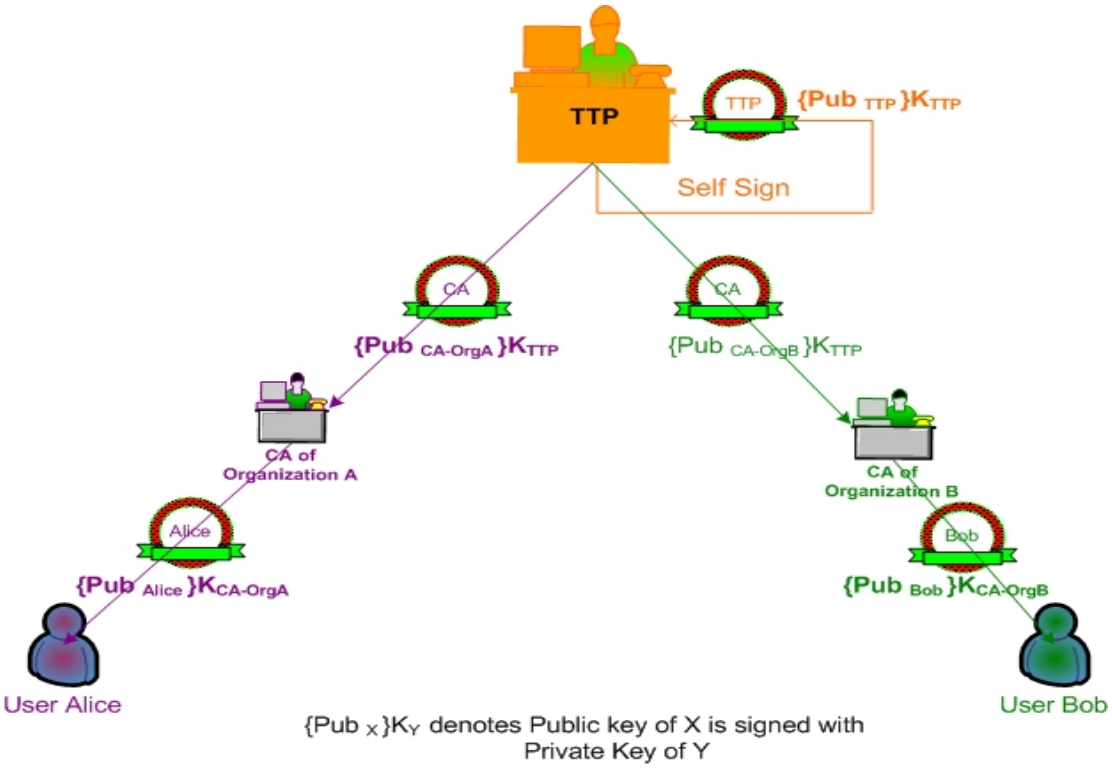

Figure 5.2: A scenario using a TTP.

User Alice has her public key certified by CA of Organization A. Similarly, User Bob has his public key certified by CA of Organization B. At the bottom of a certificate chain is the certificate issued by the CA authenticating the user’s public key. The next certificate in the chain is one that authenticates the CA's public key, which is the certificate issued by the TTP. The chain ends with the self-signed certificate of the TTP. These three certificates form the “Chain of Trust” which is indicated as a hierarchy of

certificates as shown in the figure 5.2. Each certificate in the chain (after the first) thus authenticates the public key of the signer of the previous certificate in the chain. When a service from Organization A is presented with a certificate chain from a user Bob, the service can authenticate Bob as there is a common signer among their CA and the user. The chain of trust is established here using the Java Keytool [31], which is the key and certificate management utility. It enables users to administer their own public/private key pairs and associated certificates for use in authentication. The top-level TTP certificate is self-signed. However, the trust into the TTP's public key does not come from the self-signed certificate itself (anybody could generate a self-signed certificate). The reason that the public key is stored in a certificate is because this is the format understood by most tools, so the certificate in this case is only used as a "vehicle" to transport the TTP's public key. Before the TTP certificate is accepted to create the chain using keytool, the certificate is printed and the displayed fingerprint is compared with the well-known fingerprint (securely acquiring the finger print requires out-of-band steps) for correctness.

Thus, the cross-domain authentication is achieved using a common TTP. The next section describes the approach used for authorizing actions among organizations.

5.3.2 Authorization using Broker

The procedure above achieves user authentication across multiple organizations. This however is not sufficient. A user still needs authorization information, i.e., the credential to access a service. The following sub-section details how cross-domain service access can be achieved.

5.3.2.1 Role of a Broker:

A Broker is an entity who can issue credentials on-behalf of a CA. The Broker can be the single point of contact so that the users do not have to contact CAs of multiple organizations. The CA of an organization can delegate his rights of issuing credentials to the Broker. The Broker can act as the "delegated CA". The Broker must maintain a trusted relationship with every CA of all organizations that delegates their authority of issuing credentials.

5.3.2.2 Delegating Authorization with KeyNote Trust Management

Authorization using Broker is achieved using the delegation of credentials supported by the KeyNote Trust Management. Anyone who has a valid credential can delegate his rights to another user by issuing a credential. For example, a user "Alice" from an Organization "ITTC" has access to a Nose Service from 12.00 pm to 6.00 pm on 10/07/07.

```
KeyNote-Version: 2
Authorizer: "x509-base64:MIIEBzCC.."
Licensees: "x509-base64:MIIECjC.."
Conditions:
(( app_domain == "SensorNet") && ( Provider == "ITTC" ) &&
 (Time <= "1151465644580"  && Time >= "1161465647580") &&
(ServiceID == "NicholsHallNoseService001") && (MachineName ==
"sentinel.ittc.ku.edu") && (Role == "Reader")) → "allow";
Signature: "sig-rsa-sha1-base64:XQZopw.."
```

Figure 5.3: A credential issued to user Alice to access Nose Service.

Figure 5.3 shows the credential issued to Alice. User Alice decides to delegate her rights to User David of the same organization, to access the Nose Service from 1.00 pm to 5.00 pm on 11/07/07. She creates a credential and signs with her private key. This delegated credential is shown in the figure 5.4.

```
KeyNote-Version: 2
Authorizer: "x509-base64: MIIECjC.."
Licensees: "x509-base64:MIIZAkA.."
Conditions:
(( app_domain == "SensorNet") && ( Provider == "ITTC" ) &&
 (Time <= "1151465644685"  && Time >= "1161465647980") &&
(ServiceID == "NicholsHallNoseService001") && (MachineName ==
"sentinel.ittc.ku.edu") && (Role == "Reader")) → "allow";
Signature: "sig-rsa-sha1-base64:XQZopw.."
```

Figure 5.4: A delegated credential issued from Alice to David to access the Nose Service.

David has to provide the delegated credential given in the figure 5.4 and also the credential issued to Alice given in the figure 5.3 to get authorized from the Nose Service.

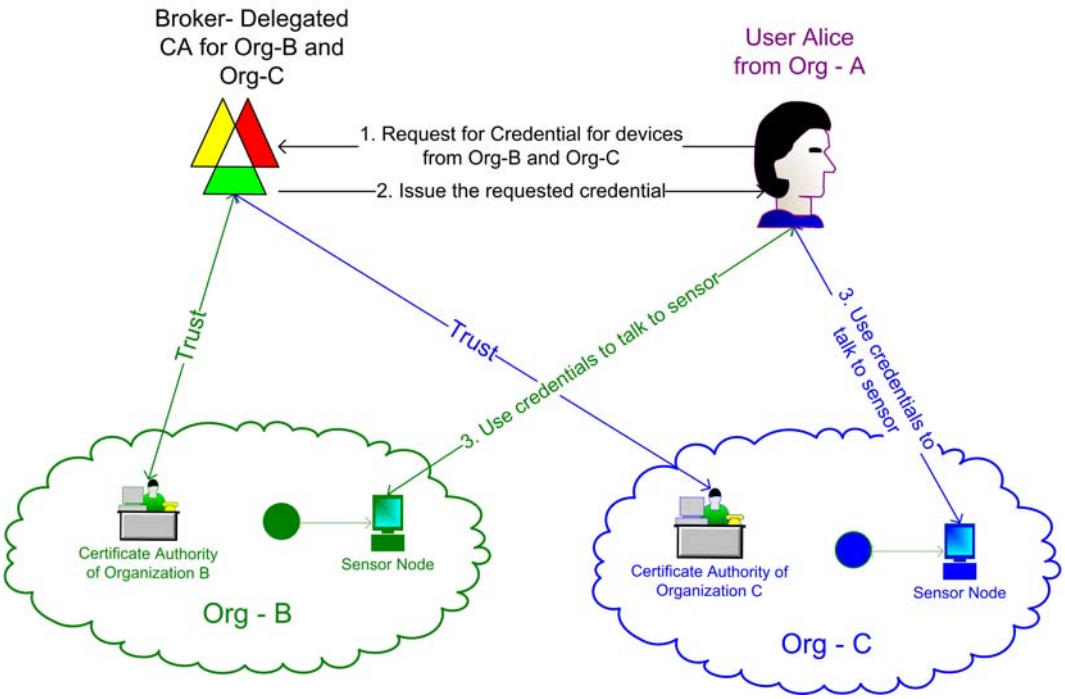

Figure 5.5: A scenario showing the use of a Broker.

The trust relationship between CA of an organization and the Broker is established by issuing a credential. The CA issues a credential to the Broker delegating his rights of issuing credentials. When a user requires a credential to talk to a device from another organization, he contacts the Broker. The Broker issues a credential to this user delegating the rights to access the device. Usually credentials stay with the users and the CA does not keep track of the issued credentials. In this case, the user has to provide the Service: 1) the credential issued to the Broker for authorizing the Broker and 2) his credential issued by the Broker. This causes difficulty in managing credentials when the user wishes to talk to devices from multiple organizations and does not provide any solution or improvements to the scalability issue discussed in the previous scenario. Hence, a copy of the credential issued to the Broker is retained by the CA and provided to any service on service startup just like the Policy authorizing the CA. This allows the Service, or particularly the KeyNote engine to know that the organization trusts the Broker as an authority, apart from their CA. The user can provide only his credential to the Service. When the user requests for access to devices from multiple organizations, the Broker can issue one credential authorizing all the requested devices. Thus, the user is required to carry only one credential sufficient to talk to devices from multiple organizations as shown in the figure 5.5. This approach simplifies the scalability issue presented in the previous scenario. The organizations can decide on the services it wishes to advertise and be used by the users of other organizations. The CA of the organization can issue credentials to the Broker only for these services, which can be delegated to users of other organization. This approach is flexible as the organization has control on what services are external and internal to the organization.

5.3.2.2.1 Limitations

The credential of a Broker provides him with a delegated authority to access services under the conditions mentioned in it. This implies that a broker becomes a valid user of the system who can access the services just as well as the CA who issued it, or any other authorized user. This condition may not however be desirable, if the Broker is required by the multi-owner architecture, not to be able to access the service himself. However such a condition cannot be represented by KeyNote as it does not have feature to provide a "meta-credential", i.e., a credential saying that an entity is authorized to issue "credentials". The delegation from the broker to a user can then be achieved by issuing credential with conditions which forms a proper subset of the conditions contained in the broker's credential. Even so, there is no mechanism within KeyNote to verify whether the conditions in a user's credential forms correct subset of those in Broker's credential during the time of issue. And a faulty credential is identified only when the credential is used for service access. Faulty credentials can be avoided if the Broker has the complete details about the list of services available, method to role mappings and the policies enforced by the organizations before issuing a credential.

5.3.2.3 Example

Consider a case where an Organizer from Organization C wishes to collect data from databases of Organization A and Organization B and maintain a database of his own. The steps that are required to achieve this case are:

Step 1: The Broker should have a trust relationship with the CA of Organization A, B and C. Each Organization issues a separate credential to the Broker representing this trust. The following figure 5.6 represents the trust relationship between the Broker and CA of Organization A, where the CA allows the Broker to delegate access to a particular Sensor Database service. The Sensor Database service is provided with this credential to indicate the KeyNote engine, that they trust the Broker apart from their CA.

KeyNote-Version: 2
Comment: CA-OrgA delegates authority to Broker for access to a particular SensorDatabase Service
Authorizer: CA-OrgA "x509-base64:MIIEBzCC.."
Licensees: Broker "x509-base64:MIIECjC.."
Conditions:
((app_domain == "SensorNet") && (Provider == "Organization-A") &&
(Time <= "1151465644580" && Time >= "1161465647580") &&
(ServiceID == "SensorDatabase002"))→ "allow";

Figure 5.6: Trust relationship between CA-OrgA and Broker.

Similarly, the Organization B provides a credential to the Broker which is shown in figure 5.7. Figure 5.7 shows that the Broker is allowed to delegate "Reader" privileges on all the Database services provided by this organization.

KeyNote-Version: 2
Comment: CA-OrgB delegates authority to Broker for "Reader" access to all Database services.

Authorizer: CA-OrgB “x509-base64:MIIEBzCC..”
Licensees: Broker "x509-base64:MIIECjC..”
Conditions:
((app_domain == “SensorNet”) && (Provider == “Organization-B”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == "^.*Database.*$") && (Role == “Reader”))→ “allow”;

Figure 5.7: Trust relationship between CA-Organization B and Broker.

Step 2: The Organizer contacts the Broker and requests the Broker to issue a credential to read Sensor information from the databases owned by Organization A and B. The Broker can issue one credential meeting the requirements of the Organizer. Figure 5.8 shows the credential that can be used in such a case.

KeyNote-Version: 2
Authorizer: Broker "x509-base64:MIIEBzCC..”
Licensees: Organizer"x509-base64:MIIECjC..”
Conditions:
((app_domain == “SensorNet”) && (Provider == “OrganizationA”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == “SensorDatabase002”) && (Role == “Reader”)) → “allow”;

((app_domain == “SensorNet”) && (Provider == “Org-B”) &&
(Time <= “1151465644580” && Time >= “1161465647580”) &&
(ServiceID == " SensorDatabase001") && (Role == “Reader”)) → “allow”;
Signature: "sig-rsa-sha1-base64:XQZopw..”

Figure 5.8: A credential issued by the Broker where the Organizer can read from Sensor Databases of Organization A and B.

Step 3: The Organizer contacts the Database Services and provides this credential for authorization.

The above scenarios represent the possible situations in a practical multi-owner application. In order to understand the implementation issues that might arise to deploy a working model, a prototype was implemented. The following chapter discusses the prototype in detail.

6 Prototype Implementation

A prototype was developed to demonstrate and verify the proof-of-concept of the proposed access control framework for the multi-owner architecture. It required identifying different functional components as various implementation modules. This chapter describes the various components of the prototype.

6.1 Overview

Figure 6.1 shows the prototype of the architecture that was implemented.

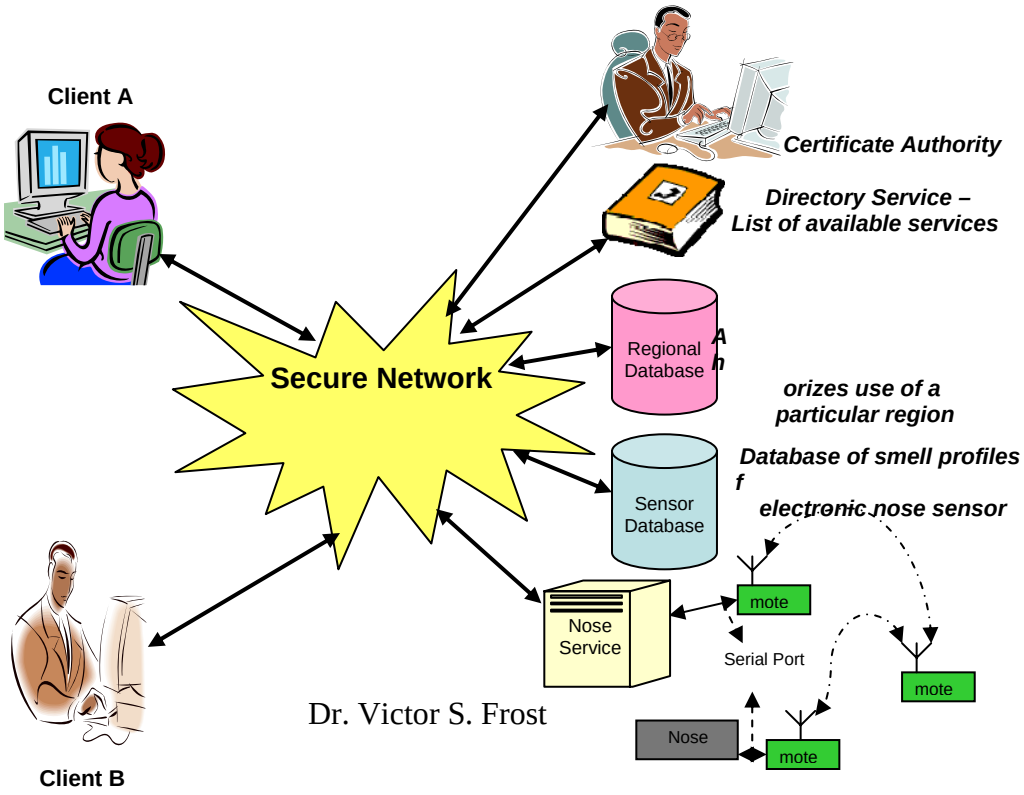

Figure 6.1: An architecture for Prototype implementation.

The tasks that were required to demonstrate the proof-of-concept of the architecture include:

- Device Layer: Selecting a sensor and developing a sample sensor service controlling the sensor. The sensor chosen was Cyranose as it was easy to program with sufficient features to demonstrate access control in a multi ownership environment. The service chosen was termed the Nose Service to be consistent with the service it provides. The Nose Service controls the Cyranose by downloading smell profiles which represents the control of Sensor Service on the Sensor. The Cyranose replies with the result of the actions it performs which represents the response given by the Sensor as given in the Section 2.2.1 of the multi-owner Architecture.
- Repository Layer: Developing a sample Sensor Database, Directory Service and Regional Database as given in the Section 2.2.2 of the multi-owner architecture. We chose to use the Service Directory and Regional Database provided by the ACE infrastructure. We chose to establish a Sensor Database of smell profiles used by the Cyranose to demonstrate the functionalities of a Sensor Database.
- Developing a client to talk to the sensor service and sensor database. We chose to develop a Nose Client program to communicate with the Nose Service and Sensor Database Service to demonstrate the functionality of the "Collector" between the Device and the Repository layer as given in the Section 2.2.1.

- Application Layer: Setting up CA to demonstrate the distribution of public key certificates and credentials as given in the Section 2.2.3.
- Demonstrating the access control mechanism in the transactions between the client, the database and the sensor service using the KeyNote Trust Management system.

6.1.1 Nose Service

A nose service was written in the ACE framework to communicate with the Cyranose. An IEEE 1451 NCAP (Network Capable Application Processor) Server [32] was developed and then used to talk to the nose, send and receive commands through a mote network. The Nose service was designed in such a way that this NCAP server is a private member of this service so that no intruder can take control of this NCAP server to talk to the nose. This service had just one method called "execute" to be consistent with the IEEE 1451 sensor specific standard. The Cyranose was thus made to be IEEE 1451 complaint.

The execute method takes the following parameters:

1. OpId – Operation id, id of the method that has to be executed in the NCAP Server. The methods supported are:
    a. Load Profile : Load a new smell profile from the sensor database to the nose
    b. Start Identification : Start a new identification in the nose
    c. Fetch Results: Retrieve the results of the last identification from the nose.
2. Mode – the Operation Mode of the sensor
3. ArgArray – A container class of input and output parameters such as the profile to the loaded and results of the actions (Load Profile, Start Identification and Fetch Results) used in communicating with the NCAP Server.

6.1.2 Nose Client

A Nose Client program was written to provide a user interface to interact with the Nose service (connected to a nose device). It provides the user interface components to select a profile, load the profile to the device, start a new identification in the device, and view the result of the last identification.

6.1.3 Database Service

A Sensor Database was created to store smell profiles. The corresponding Sensor Database service was also written to facilitate interaction with the database.
The database consists of a table with fields:

1. Id : Name of the smell profile (e.g., Coca cola or Isopropyl alcohol)
2. Profile : Actual smell profile corresponding to the Id.

The Sensor Database service supported the following operations:

1. Adding a new profile,
2. Updating an existing profile,
3. Fetching all the profiles from the database, and
4. Deleting all the profiles from the database.

6.1.4 Configuring a Certificate Authority

The administrator “ace” was chosen to be the CA for the system. This CA was setup to issue certificates and credentials within the environment. The CA currently resides in the “ace” home directory. The CA was setup using a script (setupCA.sh). A new OpenSSL directory “CA” was created.

6.1.5 Configuring Users for the Sensor Network

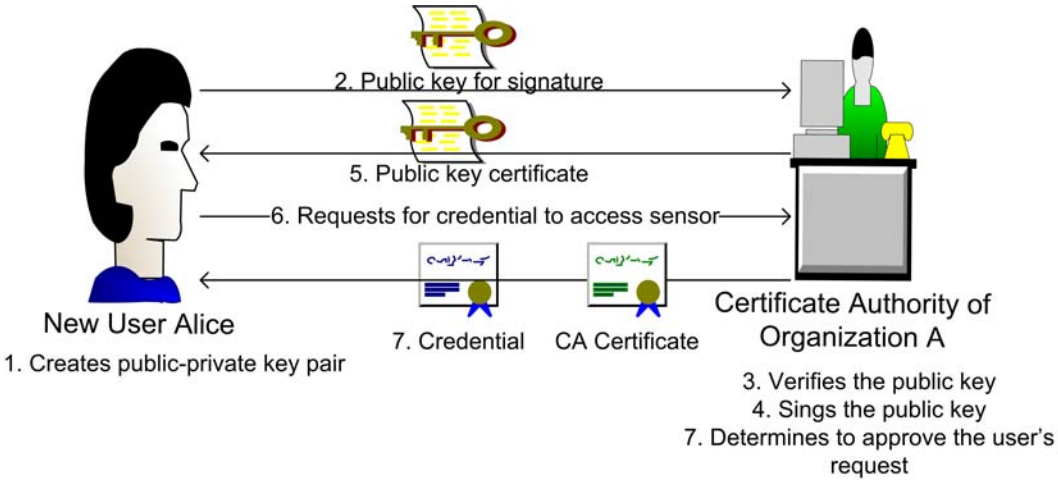

Figure 6.2: A CA issuing a public key certificate and a credential to a new user.

Each user created his public-private key pair using the KeyNote keygen tool. The keys were rsa-base64 of size 2048 bits. A certificate request was generated to request the CA to sign the public key. The CA verified this request and signed the public key. This signed public key certificate was given back to the user along with the CA certificate as discussed in Chapter 5.1. The user then imported the signed certificate and the CA’s certificate into the key rings for use under TLS and KeyNote using the Java keytool [31].

Figure 6.2 shows the interactions between a new user and the CA while placing a request for public key certificate.

6.1.6 Getting Credentials

The users in the network requested the CA for access to sensors. The CA provided the appropriate credentials using the KeyNote - command line tool for KeyNote operations.

In this current implementation, placing the credential request, verification and issuing the credential takes place offline. Figure 6.2 shows the interactions between a user and the CA to get a credential.

6.2 Testing and Results

The objective of this testing was to demonstrate the interaction of various entities: the Nose Service, the Nose Client and the Sensor Database service in a secure and controlled access environment. A demonstration portraying the access control model with two different clients, each having different rights, trying to use the Nose Service was given.

6.2.1 System Configuration

Figure 6.3 shows the system configuration that was used for this demonstration. The following steps describe the demonstration procedure:

1. The Service Directory, the Regional Database service and the Sensor Database service were executed on one computer (khan.ittc.ku.edu - Pentium III 750 MHz processor, 256 MB RAM, 10 GB HDD, Red Hat Enterprise Linux WS release 4).
2. Nose service was connected to the Cyranose through a multi-hop (wireless) Mote network on one computer (dhcp059.ittc.ku.edu - Dell Latitude D820, Pentium Dual Core T2500 2.0 GHz, 1 GB RAM, 80GB HDD, Fedora Core 5 (2.6.16)).
3. Nose client program was executed on one computer (halflife.ittc.ku.edu - Pentium III 933 MHz processor, 512 MB RAM, 80 GB HDD, Red Hat Enterprise Linux WS release 4).
4. User "A" ("satyam") was given individual credentials to contact the Service Directory, Regional Database, Nose Service and Sensor Database. He was given permissions to "Load a Profile" with the Nose Service but not to "Start the identifications" or "View results". Figure 6.4 shows the credential given to contact the Nose Service:

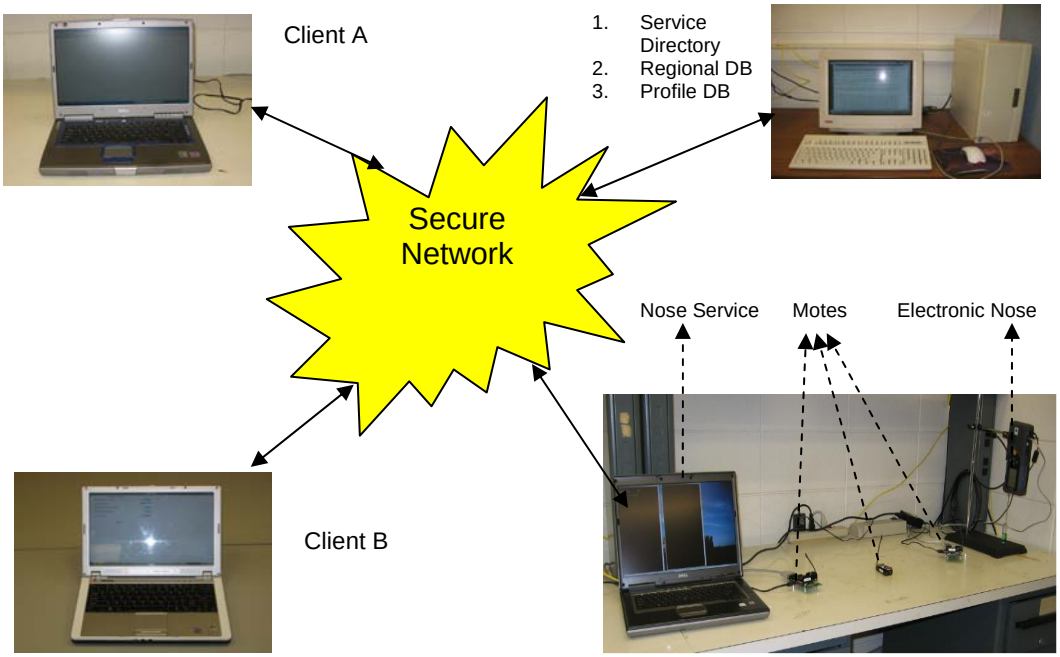

Figure 6.3: Configuration used for the demonstration of access control with two clients with different permissions.

---

keynote-version: 2
authorizer:"x509-base64:MIIEBzC …. "
local-constants:KEY1="x509base64:MIIECj…"
licensees: KEY1

conditions: ((APP_DOMAIN == "SensorNet") && (time >= "1152287101810") && (time <= "1183823100000") && (opid == "P_NOSEFBLOCK_LOAD_PROFILE")) -> "true";
signature: “sig-rsa-sha1-base64:XQ..”

Figure 6.4: Credential given to “User A” to contact Nose Service to load a profile.

Figure 6.5 shows the credential given to contact the Sensor Database.

Keynote-version: 2
authorizer:"x509-base64:MIIEBzC …. "
local-constants:KEY1="x509base64:MIIECj…"
licensees: KEY1
conditions: ((APP_DOMAIN == "SensorNet") && (time >= "1152287101810") && (time <= "1183823100000")) -> "true";
signature: “sig-rsa-sha1-base64:XQ…”

Figure 6.5: Credential given to “User A” to contact the ServiceDirectory.

5. Another user “B” (“mpradeep”) was given credentials to contact the Service Directory, Regional Database and Nose Service. He was given permissions to perform “**Start Identification**” and “**View the results**” in the sensor, but did not have rights to “Load a new profile” in the sensor. Figure 6.6 shows the credential given to user “B”.

keynote-version: 2
authorizer:"x509-base64:MIIEBzC …. "
local-constants:KEY1="x509base64 MIIECjCC…"
licensees: KEY1
conditions: ((APP_DOMAIN == "SensorNet") && (time >= "1152287101810") && (time <= "1183823100000") && (opid == " P_NOSEFBLOCK_FETCH_RESULT" || opid == "P_NOSEFBLOCK_START_IDENTIFICATION)) -> "true";
signature: “sig-rsa-sha1-base64:XQ..”

Figure 6.6: Credential given to “User B” granting access to start a new identification and view the results with the Nose Service.

6.2.2 Results

Test Case 1 - User can perform only load profile:

User A used the Nose Client program and contacted the Sensor Database. He selected the profile for “Isopropyl alcohol” and contacted the Nose Service to load this profile to the nose. This action was performed successfully as the User A had permission to load a profile. When user A tried to perform “Start Identification or Fetch Results”, he was denied access, as his credentials did not match the action requested. Figures 6.7 and 6.8 show the interactions performed by user A with Profile Database and Nose service.

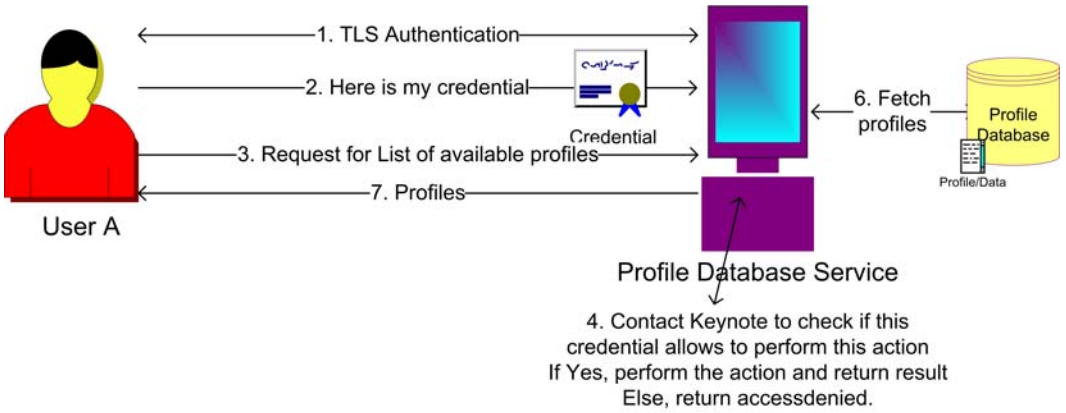

Figure 6.7: Test case where User A selected a profile from Sensor database.

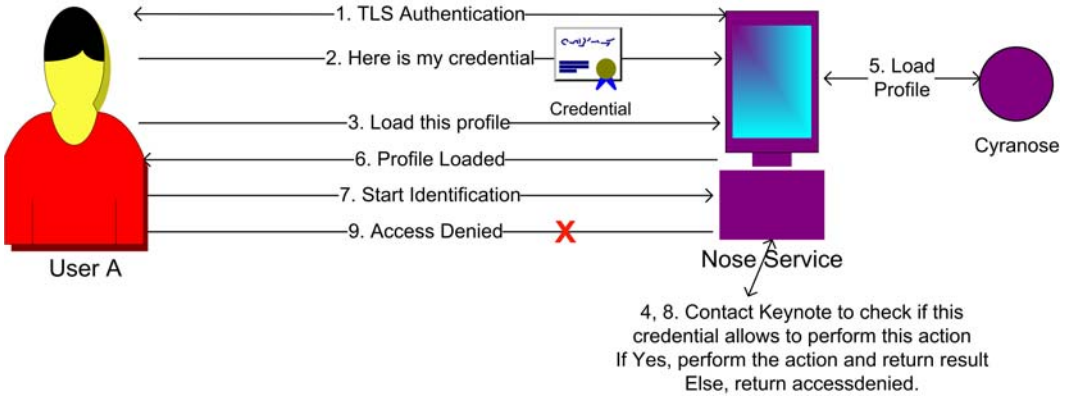

Figure 6.8: Test case where User A was granted access for loading a profile and denied access for start identification.

Test Case 2 – User can perform start identification and fetch results:

User B requested the Nose Service to start a new identification and this operation was granted access as he had credentials to perform this action. User B could not contact the Sensor Database service as he did not have credentials to talk to this service. He requested the Nose service for loading a dummy profile and was denied access as shown in the figure 6.9.

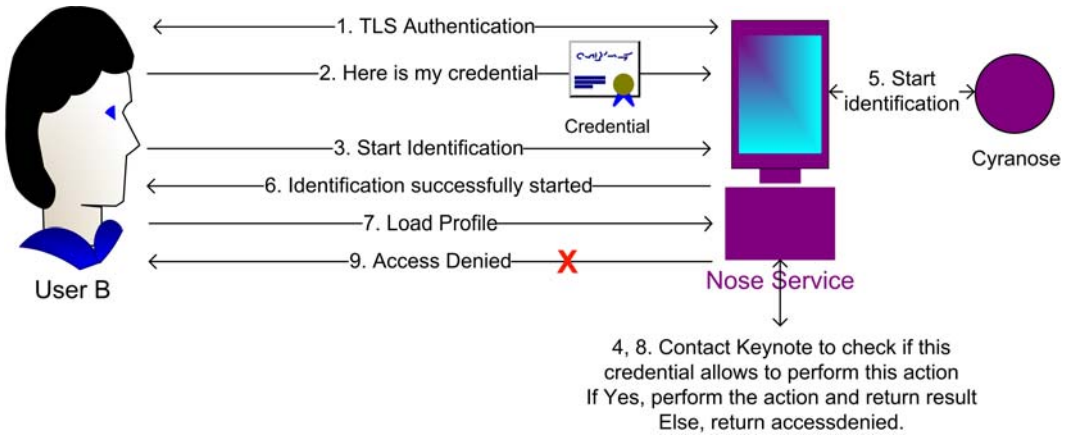

Figure 6.9: A test case where User B could perform start a new identification but could not load a profile.

The demonstration to show access control was successful. The communication between the Nose Client, Nose Service and the Sensor Database service was shown. User with valid credential was granted access and user with invalid credential was denied access.

6.2.3 Lessons Learned

Some of the lessons learned from the demonstration include:

1. While testing the access control using the credentials, the same credential that was used to talk to the Service Directory worked with Sensor Database service. We learned that the credentials lacked the capability to distinguish between the services. In the multi-owner architecture, it is possible for two sensor services to provide the same function. With the limited set of attributes such as APP_DOMAIN, time, method provided by the ACE environment, the credential could distinguish services and method for a specific service. We learned that the credentials have to be extended to be more expressive.
2. The set of action attributes provided by ACE environment were embedded within the services. We learned that it was necessary to find a mechanism to externally control, add or delete a few set of action attributes.
3. There were four different credentials, one for each of the services Service Directory, Regional Database, Sensor Database and Nose Service. It would be difficult for a user to manage multiple credentials. We learned that with KeyNote it was possible to combine multiple credentials into one by providing more clauses in the conditions. For example, figure 6.10 shows a single credential with two clauses in the conditions, one for contacting the nose service and one for contacting the Service Directory to see the list of services available.

```
keynote-version: 2
authorizer:"x509-base64:MIIEBzC …. "
local-constants:KEY1="x509base64 MIIECjCC…"
licensees: KEY1
conditions:
((APP_DOMAIN == "SensorNet") && (time >= "1152287101810") && (time <=
"1183823100000") && (opid == " P_NOSEFBLOCK_FETCH_RESULT" ||  opid ==
"P_NOSEFBLOCK_START_IDENTIFICATION)) -> "true"; #clause-1

((APP_DOMAIN == "SensorNet") && (time >= "1152287101810") && (time <=
"1183823100000") && (method == “getServices”)) -> "true"; #clause-2
        signature: “sig-rsa-sha1-base64:XQ..”
```

Figure 6.10: An example of a single credential with multiple clauses.

4. We learned that the Service Author and the Credential Author need to agree upon a standard nomenclature to provide values for attributes such as “OpID” in the credential. It was necessary to come up with a set of action attributes and their nomenclature that can be used for the multi-owner architecture.

5. We learned that any computer that has to register with the Service Directory on a different computer has to start the RMI registry on its own. We learned to start RMI Registry before starting any service.

6.3 Credential Extensions

The lessons learned from the previous section pointed to the need to extend the set of action attributes in ACE to address the large set of devices and actions. Chapter 4 described the limitation of the set of action attributes used in ACE and the extension made to that set to develop a core set of attributes for the multi-owner environment. This section describes the prototype of the credential extension and the tests performed to demonstrate this extension.

6.3.1 System Configuration

The following configuration was set up to demonstrate the credential extensions:

1. Nose Service was chosen as the sensor service for testing. The nose device was directly connected to one computer (khan.ittc.ku.edu - Pentium III 750 MHz processor, 256 MB RAM, 10 GB HDD, Red Hat Enterprise Linux WS release 4) using a serial port.
2. The Nose service was started by the “ace” administrator. The Nose Service was given the *ServiceID "NicholsHall_Nose001”* by providing an external action attribute file.
3. The Nose client was executed by user “satyam” on one computer (halflife.ittc.ku.edu - Pentium III 933 MHz processor, 512 MB RAM, 80 GB HDD, Red Hat Enterprise Linux WS release 4).

Figure 6.11 shows the configuration used for credential extensions.

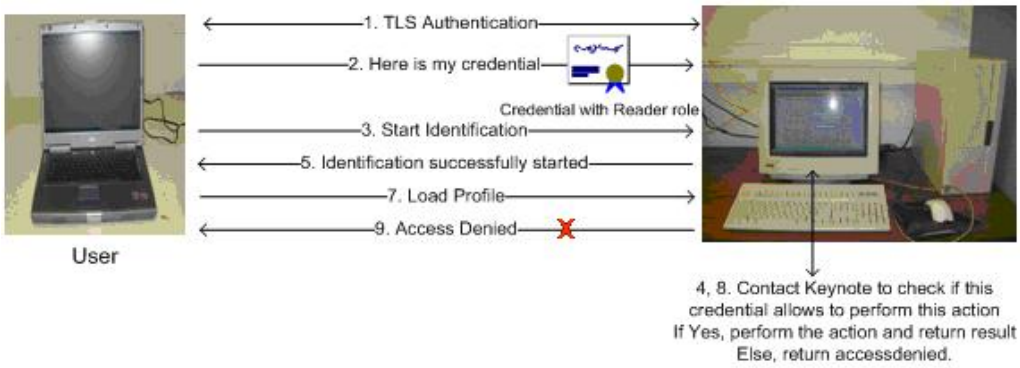

Figure 6.11: Setup used for demonstrating credential extensions.

6.3.2 Testing and Results

The user was given multiple credentials. To demonstrate the expressiveness and the granularity in the credentials, the credential to the Nose service was changed in each test case to show the different possibilities of access control.

Test Case 1: A User starts a new identification and view the results but cannot load a profile:

In this test case, the user could only start a new identification and see the result of the last identification in the electronic nose, and could not perform any other action as given in the figure 6.12.

```
KeyNote-version: 2
Authorizer: "x509-base64:MIIEBzC…"
Licensees: "'x509-base64:MIIECjCCA3OgAw.."
Conditions: ((APP_DOMAIN == "SensorNet") && (Time >= "1152287101810") &&
(Time<="1183823100000")&&(ServiceClassHierarchy==
"SecureUnicastServer.Base.Service.Device.SensorNetwork.ChemicalSensor.Nose")&&
((Method == "execute") && (FirstArgValue == "P_NOSEFBLOCK_FETCH_RESULT"
||  FirstArgValue == "P_NOSEFBLOCK_START_IDENTIFICATION"))) -> "allow";
Signature:
        "sig-rsa-sha1-base64:KISi0RAbUlYw1z+dr6oOd8hMagAP\
        1nQ2IBgVMNJDenZ1H…"
```

Figure 6.12: A test case where the user could start a new identification, view the result but could not load a profile.

Since the method "execute" in the Nose Service is a special method, it does not have any role associated with it. Hence the credential specifies the method name explicitly. The first argument value identifies the operation that the "execute" method needs to perform.

When the client requests for "Start Identification", the identification was started and the result was show in the result box. Similarly when the client requests for "Fetch result", the result of the last identification was fetched from the nose and displayed in the result box. If the user tried any other action, he/she was denied access.

Test Case 2: User has role "Reader" and "Modifier" on one sensor:

In this test case, the client can modify only one nose service identified by the "ServiceID" attribute. The client can switch it ON or turn it OFF according to its current power condition, and see the result of the action. Figure 6.13 shows the credential provided to the nose service.

```
KeyNote-version: 2
Authorizer: "x509-base64:MIIEBzC…"
Licensees: "'x509-base64:MIIECjCCA3OgAw.."
Conditions: ((APP_DOMAIN == "SensorNet") && (Time >= "1152287101810") &&
(Time <= "1183823100000") && (ServiceID=="NicholsHall_Nose001")&& (Role ==
"Modifier" || Role == "Reader")) -> "allow";
Signature:
        "sig-rsa-sha1-base64:KISi0RAbUlYw1z+dr6oOd8hMagAP\
        1nQ2IBgVMNJDenZ1H…"
```

Figure 6.13: A test case where the user has "Reader" and "Modifier" role on the nose.

The methods “powerOn” and “powerOff” which turns the device ON and OFF, require the role “Modifier”. The method “getPowerState” in the Nose Service which tells the current power state of the device requires the role “Reader”. Hence this client had both “Reader” and “Modifier” role on the device. In the Nose Client, the client requested for the “Current Power State” and the current power state was provided in the result box. According to the current power state, the client switched the device ON/OFF and the result of the actions was shown in the result box. This client was not authorized to perform any other method such as “execute” or “resetDevice” and hence the request for these actions was denied access.

Test case 3: User has role “Modifier” on all chemical sensors:

In this test case, the user was a Modifier for all chemical sensors.

```
KeyNote-version: 2
Authorizer: "x509-base64:MIIEBzC…”
Licensees: “"x509-base64:MIIECjCCA3OgAw..”
Conditions: ((APP_DOMAIN == "SensorNet") && (Time >= "1152287101810") &&
(Time <= "1183823100000") && (ServiceClassHierarchy ~= "^.*ChemicalSensor.*$")
&& (Role == "Administrator" || Role == "Reader" || Role == "Modifier")) -> "allow";
Signature:
        "sig-rsa-sha1-base64:KISi0RAbUlYw1z+dr6oOd8hMagAP\
        1nQ2IBgVMNJDenZ1H…”
```

Figure 6.14: A test case where the user has “Modifier” role on all the chemical sensors.

Figure 6.14 shows the expressiveness in the credential. The credential is small and meaningful and uses the wild card characters to specify all the chemical sensors instead of listing the chemical sensors one by one.

These test cases demonstrated the expressiveness and the simplicity in the credentials.

The credentials can provide accesses to a client from a broader level to a very narrow or precise control on the capabilities of the devices. The attribute “Role” helps to avoid listing all the methods that a client wishes to perform making the simple, small and easy to understand.

6.4 Cross-Domain Communication

The previous section demonstrated the extensions to the ACE set of action attributes to form the core attributes to achieve the granularity required for multi-owner architecture. Chapter 5 discussed the additional challenges arise when more than one organization is involved with regards to authentication of identity and authorization of actions. Section 5.3 proposed a solution of using a TTP (for authentication) and Broker (for authorization). This section details the prototype and the experiments conducted to verify the prototype.

In order to demonstrate the cross-domain authentication, the following tasks were taken:

1. A new “TTP” was created. The TTP keys were generated and self-signed certificate of the TTP was created using OPENSSL.
2. Three Organizations A, B and C were chosen for the experiment. The CAs of Organization A and B have a trust relation with the TTP. Hence, the public keys of CA-Org-A and CA-Org-B are certified by the TTP. The TTP must indicate that these two CAs are not just end users, but authorities to certify other users in their organization. The TTP includes the X.509 extensions indicating this requirement, which is given below: *basicConstraints=CA:TRUE,pathlen:1*
   The pathlen parameter indicates the maximum number of CAs that can appear below the TTP in a chain. If there is a CA with a *pathlen* of zero, it can only be used to sign end user certificates and not further CAs. In this case, there are no sub-domains within the organizations, and hence the *pathlen* is 1. But the number can be modified to include sub-domains within the main organizations. Java Keytool is used to manage a keystore (database) of the certificates. The certificates are protected using the keystore_password [31]. The certificate chain for CA-Org-A is shown in the figure 6.15.

CA-OrgA $ /tools/java/i586/jdk1.5.0_09/bin/keytool -v -list -keystore keystore
Enter keystore password:

Keystore type: jks
Keystore provider: SUN
Your keystore contains 2 entries

Alias name: ca-orga-cert
Creation date: Oct 30, 2007
Entry type: keyEntry
Certificate chain length: 2
Certificate[1]:
Owner: EMAILADDRESS=ca-org-A@ittc.ku.edu, CN=ca-org-A, OU=ITTC, O=KU, L=Lawrence, ST=KS, C=US
Issuer: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Serial number: 1
Valid from: Sun Oct 28 19:40:25 CDT 2007 until: Mon Oct 27 19:40:25 2008
Certificate fingerprints:
MD5: 8A:AF:22:BE:A2:8C:3C:F3:DD:18:D1:7B:86:9A:86:58
SHA1: 17:82:CE:4E:FC:00:B7:8D:F3:49:19:B1:FD:B7:A3:CB..
Certificate[2]:
Owner: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Issuer: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Serial number: 0
Valid from: Sun Oct 28 19:27:35 CDT 2007 until: Mon Oct 27 19:27:35 2008
Certificate fingerprints:

MD5: 6B:D3:10:48:9D:A1:6D:F4:3F:D3:A5:D7:DD:69:EC:AF
SHA1: 00:F4:B6:0E:3C:19:EB:8C:8A:B3:48:8E:CC:7A:2E:E6:79:3A:95:CB

Figure 6.15: Certificate chain of CA-Org-A.

The figure 6.15 shows that there are 2 certificates in this chain:

a. The public key certificate of CA-Org-A issued by the TTP, and
b. The self-signed certificate of the TTP.

CA of Organization C was created. This organization does not have a trust relationship with the TTP. Hence, the CA-Org-C has a self-signed certificate.

3. User "Alice", "Bob" and "Carol" of Organization A, B and C were created. Their public keys are certified by the CA of the respective organization. The certificate was indicated that these users are end-users and not CAs, by providing a "False" to the X.509 extensions:
basicConstraints=CA:FALSE
The certificate chain of Bob is shown in the following figure 6.16.

[bob_orgB@terbium .ace_test]$ /tools/java/i586/jdk1.5.0_09/bin/keytool -v -list -keystore keystore
Enter keystore password: i#J[^ji1s|

Keystore type: jks
Keystore provider: SUN

Your keystore contains 3 entries

Alias name: bob-cert
Creation date: Oct 30, 2007
Entry type: keyEntry
Certificate chain length: 3
Certificate[1]:
Owner: EMAILADDRESS=bobbob@orgb.com, CN=bob, OU=ORGB, O=ORGB, L=Dallas, ST=Texas, C=US
Issuer: EMAILADDRESS=caorgB@orgb.edu, CN=CAORGB, OU=CA-ORGB, O=XXX, L=Dallas, ST=Texas, C=US
Serial number: 1
Valid from: Tue Oct 30 21:33:32 CDT 2007 until: Wed Oct 29 21:33:32 CDT 2008
Certificate fingerprints:
MD5: 44:7E:7E:7C:04:E9:8C:01:B8:4B:86:79:02:29:8D:72
SHA1: C4:B6:E4:7D:D9:11:B9:3B:89:49:98:20:8A:6F:90:72:A2:91:0E:BF
Certificate[2]:
Owner: EMAILADDRESS=caorgB@orgb.edu, CN=CAORGB, OU=CA-PRGB, O=XXX, L=Dallas, ST=Texas, C=US
Issuer: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Serial number: 2

Valid from: Tue Oct 30 20:36:16 CDT 2007 until: Wed Oct 29 20:36:16 CDT 2008
Certificate fingerprints:
MD5: BB:3B:99:E4:24:86:81:FC:30:53:BF:9A:9A:A9:3E:1B
SHA1:
52:01:B2:44:CE:B4:69:EB:B7:9A:8E:6E:42:ED:AA:81:6B:C9:A1:90
Certificate[3]:
Owner: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Issuer: EMAILADDRESS=dummyttp@central.com, CN=dummyTTP, OU=TTP, O=Central, L=Dallas, ST=Texas, C=US
Serial number: 0
Valid from: Sun Oct 28 19:27:35 CDT 2007 until: Mon Oct 27 19:27:35 CDT 2008
Certificate fingerprints:
MD5: 6B:D3:10:48:9D:A1:6D:F4:3F:D3:A5:D7:DD:69:EC:AF
SHA1:
00:F4:B6:0E:3C:19:EB:8C:8A:B3:48:8E:CC:7A:2E:E6:79:3A:95:CB

Figure 6.16: Certificate chain of Bob of Organization B.

4. A Broker was created. Organizations A and B have trust relationship with the Broker by providing a credential. Organization A decides to provide the “Reader” privileges from Profile Database Service to users of other organization. Organization A issues a credential to the Broker to delegate “Reader” privileges on this service to users of other organizations.
5. Bob from Organization B requests the Broker to issue a credential to talk to Profile Database service of Organization A. The Broker issues the following credential:

```
KeyNote-version: 2
Authorizer: Broker "x509-base64:MIIEBzC…”
Licensees: Bob-Org-B “"x509-base64:MIIECjCCA3OgAw..”
Conditions:
((APP_DOMAIN == "SensorNet") && (Provider == "OrganizationA") &&
(ServiceID == "NicholsHall_ServiceDir") &&
(Role == "Reader")) -> "allow";   #ServiceDir
((APP_DOMAIN == "SensorNet") && (Provider == "OrganizationA")
&&
(ServiceID == "NicholsHall_RoomDatabase") &&
(Method == "getRoom" )) -> "allow";    #RegionalDB
((APP_DOMAIN == "SensorNet") && (Provider == "OrganizationA") &&
 (ServiceID == "NicholsHall_ProfileDatabase001") &&
 (Role == "Reader")) -> "allow";   #SensorNetDBService
Signature:  "sig-rsa-sha1-base64:KISi0RAbUlYw1z+dr6oOd8hMagAP”
```

Figure 6.17: A credential issued from Broker to Bob-Org B to access Profile Database Service of Organization A.

6.4.1 System Configuration

The following configuration was set up to demonstrate the cross-domain communication:

1) Profile Database Service was started by the CA-Org-A in one computer (khan.ittc.ku.edu - Pentium III 750 MHz processor, 256 MB RAM, 10 GB HDD, Red Hat Enterprise Linux WS release 4). The service connects to the smell Profile Database. The service was given the *ServiceID "NicholsHall_ProfileDatabase001"* by providing an external action attribute file. This service is informed of the trust relationship with the Broker.
2) Organization A has a Nose Service which is internal to the Organization. The Nose Service was started on one computer (terbium.ittc.ku.edu - Intel(R) Xeon(TM) CPU 2.66GHz, 1 GB RAM, Red Hat Enterprise Linux 5).
3) The Profile Database client was executed by user "Bob" on one computer (sentry.ittc.ku.edu - Intel(R) Xeon(TM) CPU 2.80GHz, 1 GB RAM, Red Hat Enterprise Linux –SMP).

6.4.2 Testing and Results

The following test cases were performed to demonstrate the cross-domain communication.

Test Case 1: A user can authenticate and also authorize to a service from different organization:

Bob from Organization B contacts the Profile Database service. The service authenticates Bob as there is a common signer or trust between the CAs of both the organizations. Bob provides the credential issued by the Broker. The service informs the KeyNote engine the Policy authorizing the CA-Org-A, the credential issued to the Broker authorizing to delegate access to Profile Database and the credential issued by the Broker to access Profile Database. The KeyNote determines that the user "Bob" has "Reader" privileges on the Profile Database service. Hence, the service authorizes Bob to view the smell profiles in the database. The figure 6.18 given below illustrates this test case.

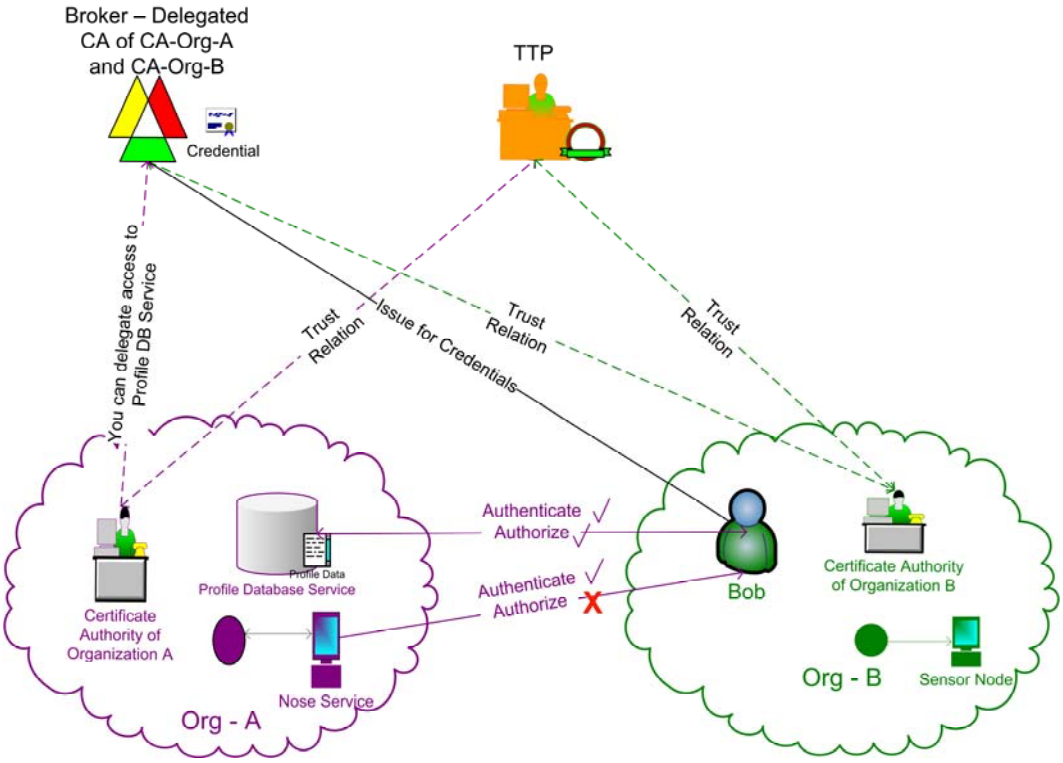

Figure 6.18: Illustration of a Cross –Domain communication showing two cases: where a user from Organization B: can 1) authenticate and authorize to Profile DB Service and 2) authenticate but not authorize to Nose Service of Organization-A.

Test Case 2: A user can authenticate but can not authorize to a service from different organization:

User Bob requests the Broker to issue a credential to contact the Nose service of Organization A. But the Nose service is internal to the organization and the Broker does not have the authority to issue credential for Nose service. Consider a case where the Broker issues a wrong credential to Bob, authorizing the access to Nose service. Bob uses this credential to contact the Nose. The Nose service authenticates Bob, as there is a common signer or trust between the CAs of both the organizations, but denies access to perform any operation on the Nose device due to the invalid credential issued by the Broker. The figure 6.18 illustrates this test case. This test case demonstrates that organization has control on deciding what services are internal and external to the organization, and even if the Broker issues a wrong credential, the service will deny access. The Broker should have knowledge of the services that are external to the organization to avoid issuing wrong credentials.

Test Case 3: A user can not authenticate and authorize to a service from different organization:

Carol from Organization C contacts the Profile Database service. The Profile Database service and Carol are not able to authenticate each other, as there is no common signer or trust between the CA of both the organizations. Hence, Carol is reported the

failure in the handshake procedure as there is no trust in the certificate chain. The figure 6.19 illustrates this test case.

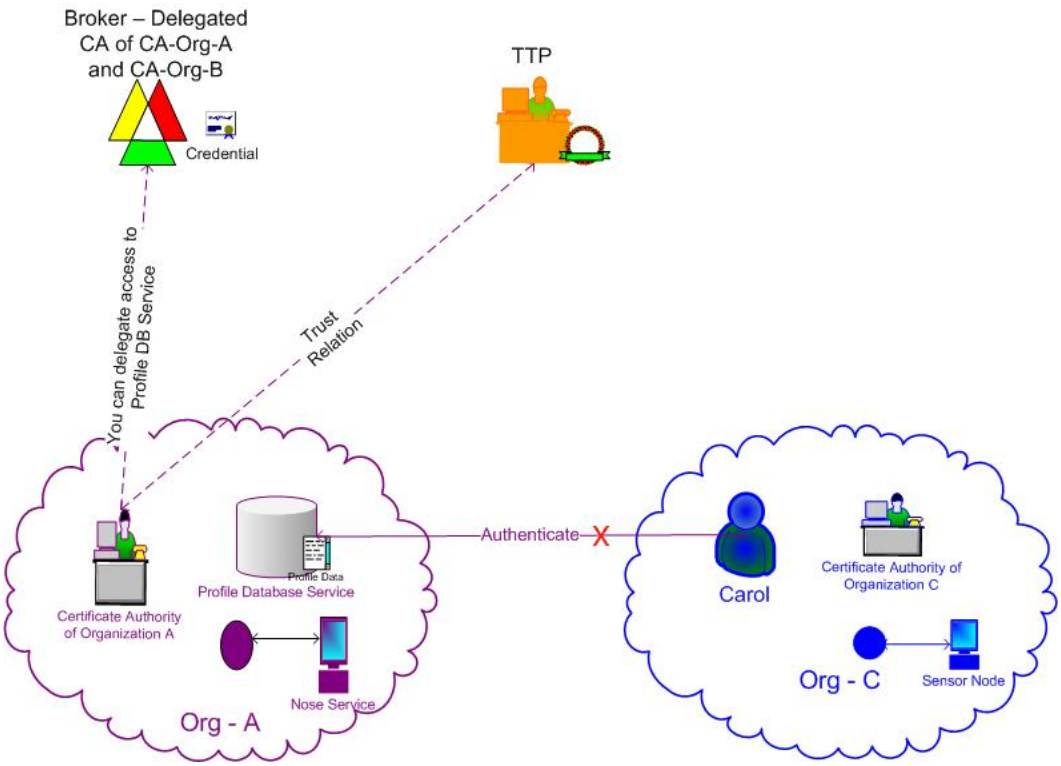

Figure 6.19: Illustration of a Cross –Domain communication where a user from Organization C cannot authenticate and hence not authorize to Profile DB Service of Organization A.

Thus, cross-domain communication is achieved. These test cases demonstrated the establishment of trusts between organizations through the TTP and Broker. The test cases also demonstrated that though there is a central authority to issue public key certificates, the control on what services to delegate and what actions that can be performed on them still remains within the organization thus providing the access-control required for multi-owner architecture.

7 Conclusion and Future Work

A unified architecture that control access to sensors by integrating very components has been proposed and studied. The developed access control framework provides for a flexible security and policy model to support the multi-owner architecture. This framework has been successfully designed and tested using a prototype implementation. Several manually implemented components of the prototype such as the offline interaction between the CA and the user for issuing credentials and the interaction between the CA and the Service Author for determining the available methods and the corresponding role mappings can be automated. The set of action attributes could be extended to include deployment specific attributes such as the physical location of the device, the connection of the device to the service etc. The architecture could be extended to support mobile sensors. Performing actions on sensors using serialized java objects can be re-implemented using a method like XML-RPC [33] to make the system accessible to

all programming languages and across all operating systems. Experiments to measure performance, scalability and deployment issues to complement prototype demonstrations could be designed.

8 References

[1] A. Hac, “Wireless Sensor Network Designs,” Wiley & Sons, West Sussex, England, 2003.

[2] D. Estrin, D. Culler, K. Pister, and G. Sukhatme, “Connecting the Physical World with Pervasive Networks,” IEEE Pervasive Computing, pp. 59–69, January–March 2002.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “Wireless Sensor Networks: A Survey,” Computer Networks, Vol. 38, pp. 393–422, 2002.

[4] SensorNet Project, Oak Ridge National Laboratory, http://www.sensormag.com/, April 2005, Vol 22. No. 4.

[5] J. M. Rabaey et al., PicoRadio Supports Ad Hoc Ultra- Low Power Wireless Networking,IEEE Comp. Mag., 2000, pp. 4248.

[6] J. M. Kahn, R. H. Katz, and K. S. J. Pister, Next Century Challenges: Mobile Networking for Smart Dust, Proc. ACM MobiCom ’99, Washington, DC, 1999, pp. 27178.

[7] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, Energy-Efficient Communication Protocol for Wireless Microsensor Networks, IEEE Proc. Hawaii Int’l. Conf. Sys. Sci., Jan. 2000, pp. 110.

[8] W. Su and I. F. Akyildiz, A Stream Enabled Routing (SER) Protocol for Sensor Networks, to appear, Medhoc- Net 2002, Sardegna, Italy, Sept. 2002.

[9] G. J. Pottie and W. J. Kaiser, Wireless Integrated Network Sensors,Commun. ACM, vol. 43, no. 5, May 2000, pp. 551-58.

[10] SensorNet Architecture Forum, August 13-14, 2003, ITTC, University of Kansas, http://www.ittc.ku.edu/workshops/sensornet/.

[11] Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J.D.Tygar. Spins: Security protocols for sensor networks. Wireless Networks, 8:521, 534, 2002

[12] Adrian Perrig, Ran Canetti, J.D. Tygar, and Dawn Song, Efficient authentication and signing of multicast streams over lossy channels, In IEEE Symposium on Security and Privacy, May 2000.

[13] Bhrat Patel and Jon Crowcroft. Ticket based service access for the mobile user. In Third annual ACM/IEEE international conference on Mobile computing and networking, pages 223233, Budapest Hungary, September 1997

[14] David W. Carman, Peter S. Kruus, and Brian J. Matt. Constraints and approaches for distributed sensor network security. NAI Labs Technical Report #00-010, September 2000.

[15] Secure Microcontrollers for SmartCards, www.atmel.com/atmel/acrobat/1065s.pdf.

[16] iButton: A Java-Powered Cryptographic iButton, www.ibutton.com/ibuttons/ java.html.

[17] J. Mauro, “Security Model in the Ambient Computational Environment,” master’s thesis, Dept. of Electrical Eng. & Computer Science, The University of Kansas, 2004.

[18] Sun Microsystems, “Java Remote Method Invocation (Java RMI)”, http://java.sun.com/products/jdk/rmi/.

[19] T. Dierks, C. Allen, “The TLS Protocol Version 1.0”, RFC 2246, January 1999.

[20] B. Kaliski, “PKCS #1: RSA Encryption Version 1.5”, RFC 2313, March 1998.

[21] D. W. Kravitz, “Digital signature algorithm,” U.S. Patent 5,231,668,1993.
[22] R. Housley,W. Polk, W. Ford, D. Solo “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, RFC 3280, April 2002.
[23] National Institute of Standards and Technology (NIST), “Specification for the Advanced Encryption Standard (AES)”, FIPS-197, November 26 2001.
[24] United States Department of Commerce, National Bureau of Standards, *Data Encryption Standard*, Federal Information Processing Standards (FIPS) Publication no. 46, January 15, 1977.
[25] M. Blaze, J. Feigenbaum, J. Ioannidis,A. Keromytis, “The KeyNote Trust-Management System Version 2”, RFC 2704, September 1999.
[26] J. Kohl, J., C. Neuman, C., The Kerberos Network Authentication Service (V5), IETF RFC 1510. 1993. http://www.ietf.org/rfc/rfc1510.txt.
[27] J. Trostle, I. Kosinovsky, and M. M. Swift. Implementation of Crossrealm Referral Handling in the MIT Kerberos Client. In Proceedings of the 2001 Network and Distributed System Security Symposium (SNDSS), pages 109.124, February 2001.
[28] Charlie Kaufman, Radia Perlman and Mike Speciner, Network Security, Second Edition, Prentice-Hall PTR, 2002.
[29] A. Jones, R. Lipton, and L. Snyder, “A Linear Time Algorithm for Deciding Security,” *Proc. 17th Annual Symp. on the Foundations of Computer Science* (Oct. 1976), 33-41.
[30] Amon Ott, Stanislav Ievlev, and Heinrich W. Klöpping. The Rule Set Based Access Control (RSBAC) Linux Kernel Security Extension. *In Proceedings of the 8th International Linux Kongress, November 2001.*
[31] Sun Microsystems, “Java keytool – Key and Certificate Management Tool”, http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html.
[32] IEEE P1451, Draft Standard for a Smart Transducer Interface for Sensors and Actuators Network Capable Application Processor (NCAP). Institute of Electrical and Electronics Engineers, Inc., New York, 1996.
[33] Winer, D.,”XML-RPC Specification”,January 1999, http://www.xmlrpc.com/spec.

A. Appendix

A.1 Nose Client with MVC Paradigm

To improve the Nose client program that is used to talk to the Nose Service, Model View Controller (MVC) paradigm was used. MVC is a design paradigm used by applications to need the ability to separate display from the data. The MVC pattern hinges on a clean separation of objects into one of three categories: model for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or view(s). It provides a clear distinction between the module containing the data, the model to display and the model to manage the user interactions.

The classes implemented for MVC include:

1. BaseClientModel - contains the basic functions that any client has to perform in order to get the handle to the remote service using RMI.
2. NoseClientModel – handles the data in the nose client
3. NoseClientView – handles the display of the GUI components
4. NoseClientController – handles the user interactions with the GUI
5. NoseClient – creates the model, view and the controller.

The interaction in the MVC classes of nose client can be of two types:

1. Controller providing the results to the View to update its display
2. Model providing the results to the View to update its display

A.1.1 Controller providing the results for the View to update its Display

Figure A.1 shows the interaction between the three classes when a “Start Identification” button is clicked by the user.

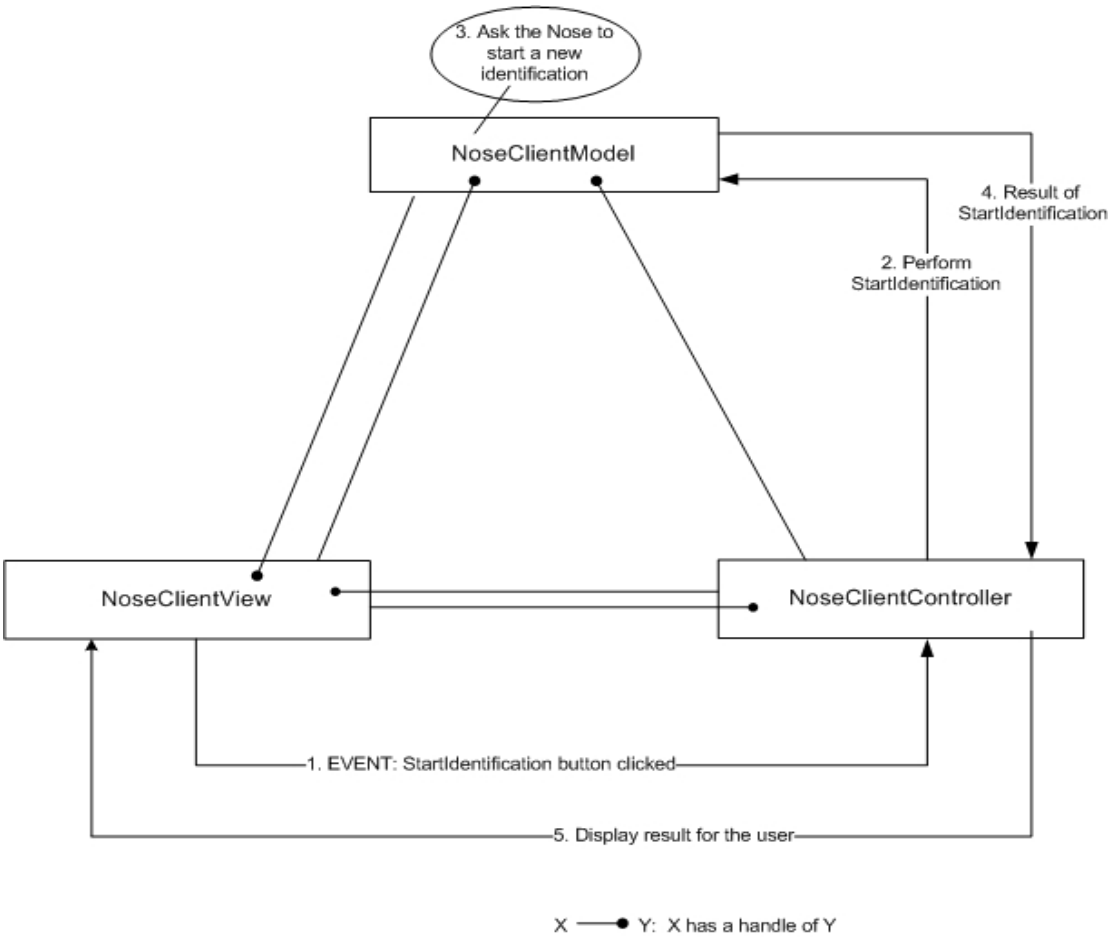

Figure A.1: MVC Communication cycle when "Start Identification" is clicked.

The sequence of operations includes:

- The User clicks the button "Start Identification"
- Controller receives this event, asks the model to perform a new identification.
- The model talks to the nose service, and asks it to start a new identification and gets the result of the action from the nose service.
- The model returns the result to the controller.
- The controller returns the results to the view to update in the result box.

A.1.2 Model providing the results for the View to update its Display

Figure A.2 shows the sequence of operations when a "Show the list of Profiles" button is clicked.

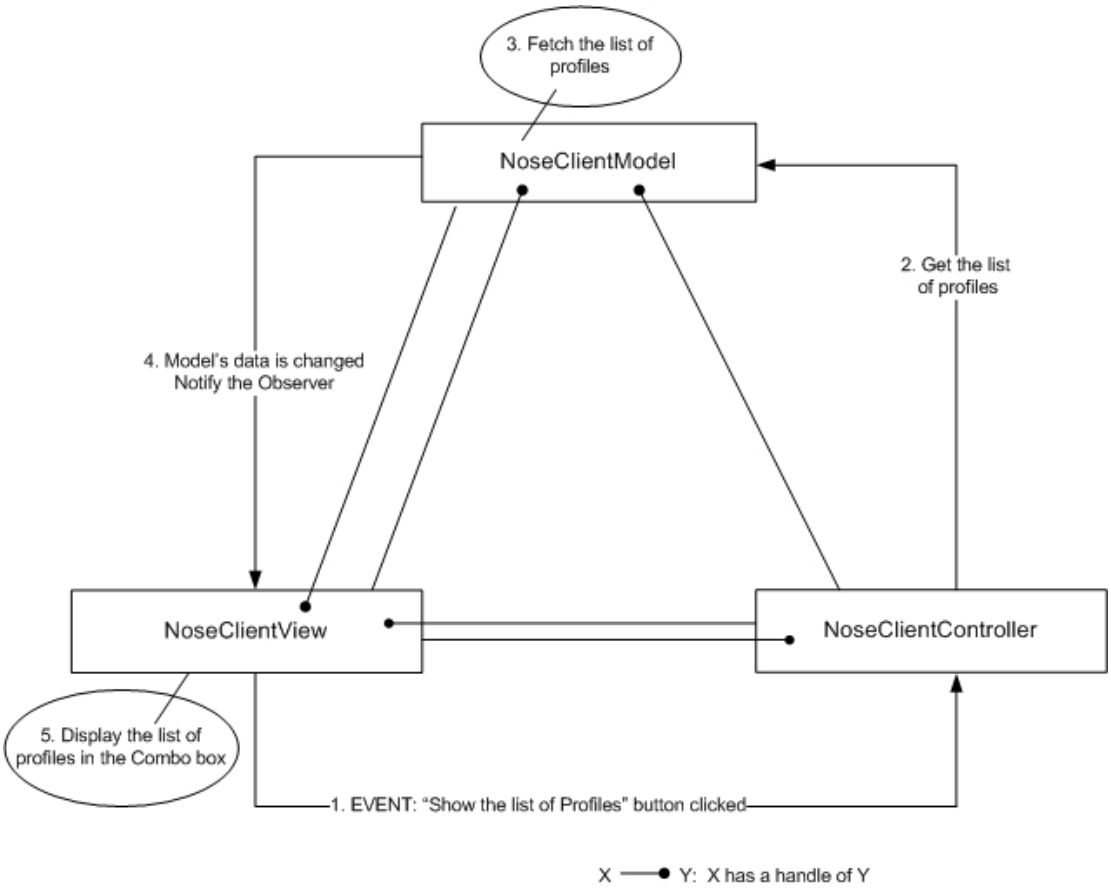

Figure A.2: MVC Communication cycle when "Show list of profiles" is clicked

The sequence of operations includes:

1. The User clicks the button "Show the list of Profiles"
2. Controller receives this event, asks the model to get the list of profiles.
3. The model does the required work to fetch the list of profiles.
4. As the model's data is changed, the model informs its "Observer" (the NoseClientView) with the new list of profiles. The view updates its combo box with the list of profiles notified by the model.