Security Architectures for Sharing of Distributed Resources in Community Based Systems

THESIS

submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

SOFTWARE ENGINEERING

by

Chavdar Bachvarov
born in Pleven, Bulgaria

Software Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Security Architectures for Sharing of Distributed Resources in Community Based Systems

by Chavdar Bachvarov

Abstract

The amount of digital assets people have grows very rapidly. This holds not only for commercial assets, such as work documents, digital images, etc., but also for a large amount of personal assets created with digital still and video cameras. An important application of these assets is sharing them with relatives, friends, colleagues, members of a sports club or other people belonging to a specific group or community. The sharing can be done while you are in close proximity to each other, by looking at the picture or video on the same device, or while at some distance from each other, by looking at it on different devices or even at different moments in time. These examples of remote sharing introduce issues with respect to the security and privacy of sharing information. How can assets be easily and securely shared with others? What are the potential concepts that can help consumers to configure easily the remote access to their personal assets?

This report targets those problems in the scope of the "Gatewayed Services" project of the Software Architectures Group in Philips Research. The project presents the concept of Gatewayed Communities deployed on a Community System somewhere on the Internet. Such a system hosts communities, where people share common assets and even services.

The project described in this paper had two phases. Initially a security architecture was evaluated for the system and a demonstrator for secure asset sharing was implemented, as an extension of the current demonstrator of the Community System. The second part of the project targets the concept of UPnP communities. An UPnP community is a distributed UPnP network, which combines the local UPnP networks of all community members. The second phase introduces UPnP Security in the security architecture of the Community System.

Committee Members:

- **Univeristy Advisor:** Leon Moonen, SE, TU Delft
- **Company Advisor:** Wim van der Linden, SwA, Philips Research Eindhoven
- **Company Advisor:** Rik Willems, SwA, Philips Research Eindhoven
- **Chair:** Arie van Deursen, SE, TU Delft
- **Member:** Georgi Gaydadjiev, CE, TU Delft
I dedicate my thesis to my father, Ludmil.
# Contents

List of Figures  ix  
Acknowledgements  xi  

## 1 Introduction

1.1 Problem description .......................... 1  
1.2 Objectives .................................... 1  
1.3 Project approach .............................. 2  
1.4 Report outline ................................. 2  

## 2 Community System Specification

2.1 The existing Community System .................. 3  
2.2 New features introduced ........................ 5  
2.2.1 Distributed asset storages .................. 5  
2.2.2 Content protection .......................... 6  
2.3 Basic Scenarios ................................ 7  
2.3.1 Device Registration ........................ 7  
2.3.2 Asset Publication ............................ 7  
2.3.3 Access to Assets ............................ 7  
2.3.4 Device Presence .............................. 8  
2.4 System Requirements ............................ 8  

## 3 Technologies Considered

3.1 Lightweight Directory Access Protocol (LDAP) .......... 11  
3.2 Kerberos ...................................... 12  
3.3 PKI ........................................... 13  
3.3.1 SSL/TLS .................................... 14  
3.3.2 HTTPS ...................................... 14  
3.3.3 IPSec ....................................... 15  
3.4 Web Services ................................... 15  
3.5 UPnP ........................................... 16  
3.6 UPnP Security ................................ 16  

## 4 Asset Community System Architecture

4.1 Security architecture selection ..................... 19  
4.2 Directory Model of the Community System ............. 20  
4.3 Conceptual view ................................ 21  
4.4 Security model .................................. 22  
4.5 Asset Community System Architecture ................ 24  
4.5.1 Community Gateway Application Server ............ 24
4.5.2 Asset Provider Applications ........................................ 26
4.5.3 Information Flow ...................................................... 27
4.6 Deployment view ......................................................... 28

5 Asset Community System Architecture realization 29
5.1 Asset Community System Application Design ......................... 29
  5.1.1 Information Model .................................................. 29
  5.1.2 Modules view ....................................................... 30
5.2 Asset Provider ................................................................... 33
  5.2.1 Asset Publisher Component ......................................... 33
  5.2.2 Asset Server Component ............................................. 34
5.3 Code organization .......................................................... 35
  5.3.1 Asset Community System Application ............................. 36
  5.3.2 Asset Provider .......................................................... 37
5.4 Project Build .................................................................. 37
  5.4.1 Community System Application .................................... 38
  5.4.2 Asset Provider .......................................................... 38
5.5 Configuration .................................................................. 39
5.6 Installation ................................................................ ....... 40

6 UPnP communities ............................................................ 41
6.1 UPnP communities - from content to services ....................... 41
  6.2 Basic Scenarios ............................................................ 41
    6.2.1 Registration of UPnP network at CS ......................... 42
    6.2.2 Discovery of UPnP Services ..................................... 42
    6.2.3 Publication of UPnP services .................................... 42
    6.2.4 Access to UPnP Services .......................................... 43
6.3 System Requirements ...................................................... 43
6.4 Community System model updated .................................... 44
6.5 Conceptual view ........................................................... 45
6.6 Security model .............................................................. 48
6.7 System Architecture ....................................................... 49
  6.7.1 Community Gateway .................................................. 50
  6.7.2 UPnP network components ........................................ 51
6.8 Deployment view ........................................................... 53

7 Related work ................................................................. 57
  7.1 SAML & Shibboleth ....................................................... 57
  7.2 VITELS ................................................................. 57
  7.3 Groove Virtual Office .................................................... 58

8 Concluding Remarks ........................................................ 59
  8.1 Conclusions ............................................................. 59
  8.2 Outlook ................................................................. 59
    8.2.1 Communities of patients ......................................... 59
    8.2.2 Virtual office ....................................................... 60
## Bibliography

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Abbreviations</td>
<td>65</td>
</tr>
<tr>
<td>B X.509 certificates</td>
<td>67</td>
</tr>
<tr>
<td>C Needham-Schroeder authentication protocol</td>
<td>69</td>
</tr>
<tr>
<td>D Asset service technical scenarios</td>
<td>71</td>
</tr>
<tr>
<td>D.1 Registration of devices</td>
<td>71</td>
</tr>
<tr>
<td>D.2 Asset publication</td>
<td>74</td>
</tr>
<tr>
<td>D.2.1 Select Assets</td>
<td>74</td>
</tr>
<tr>
<td>D.2.2 Publish assets in community</td>
<td>75</td>
</tr>
<tr>
<td>D.3 Access to assets</td>
<td>77</td>
</tr>
<tr>
<td>D.4 Device presence</td>
<td>79</td>
</tr>
<tr>
<td>E UPnP Security</td>
<td>81</td>
</tr>
<tr>
<td>E.1 Secure Device Actions</td>
<td>81</td>
</tr>
<tr>
<td>E.2 Security Console Actions</td>
<td>83</td>
</tr>
<tr>
<td>E.3 UPnP certificates</td>
<td>83</td>
</tr>
<tr>
<td>F Data Structures</td>
<td>85</td>
</tr>
<tr>
<td>F.1 Published Asset List</td>
<td>85</td>
</tr>
<tr>
<td>F.2 Offline Published Asset List</td>
<td>86</td>
</tr>
<tr>
<td>F.3 Community System Configuration file</td>
<td>86</td>
</tr>
<tr>
<td>F.4 Asset Provider Configuration file</td>
<td>87</td>
</tr>
<tr>
<td>G Glossary</td>
<td>89</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 Conceptual architecture of Community Gateway ............................................. 3  
2.2 Model of Community directories ................................................................. 4  
2.3 Implementation view of the first Gatewayed Communities demonstrator .......... 5  
3.1 Kerberos Overview ......................................................................................... 12  
4.1 Directory Model of Community System ......................................................... 20  
4.2 Conceptual view of Asset Community System .............................................. 21  
4.3 Security model of Community System ......................................................... 22  
4.4 Security architecture of Asset Community System ...................................... 24  
4.5 Community System Application Architecture ............................................. 25  
4.6 Asset Provider Architecture (modules view) ............................................... 26  
4.7 Community System Information Flow ......................................................... 27  
4.8 Community System deployment view .......................................................... 28  
5.1 Community System Application Architecture (modules view) .................... 30  
5.2 Community System Application Class Diagram .......................................... 31  
5.3 Asset Publisher Component class diagram ................................................... 33  
5.4 Asset Server Component class diagram ........................................................ 35  
5.5 Application project structure ....................................................................... 36  
5.6 Community System demonstrator for secure asset sharing code organization 36  
5.7 Asset Publisher Component code organization ......................................... 37  
5.8 Asset Server Component code organization ................................................. 37  
5.9 gws.war file structure ............................................................................... 38  
6.1 Conceptual Model of UPnP Communities ..................................................... 45  
6.2 Associations in UPnP community model ..................................................... 45  
6.3 Pre-mature conceptual view of UPnP communities ....................................... 46  
6.4 UPnP communities conceptual view ........................................................... 47  
6.5 UPnP Communities Architecture ................................................................. 49  
6.6 UPnP deployment view ................................................................................ 53  
6.7 UPnP Communities network components (deployment view) ................... 54  
6.8 UPnP Communities Invocation view ......................................................... 54  
C.1 The Needham-Schroeder authentication protocol ......................................... 69  
D.1 Device Registration Scenario ...................................................................... 73
Acknowledgements

First of all, I would like to thank everybody who contributed to the success of my master thesis. Especially, I would like to thank my supervisors Wim van der Linden, Rik Willems and Leon Moonen who guided me throughout my research work. Furthermore, I would like to express my gratitude to Jaap van der Heijden who gave me the opportunity to carry out my master thesis in the Software Architecture (SwA) group at Philips Research. Special thanks to Mariana Simons-Nikolova, Eugene Ivanov, David Simons and the rest of my colleagues at SwA for the interesting and fruitful discussions that enlarged my technical outlook.

I also give credits to my relatives and friends in Bulgaria, the Netherlands and all over the world for the support and the fun we have had together. Special thanks to my girl friend Paulina for her patience and love, especially during the years of my master study.

Finally, with deepest love and gratitude I would like to thank my parents, Tsvete and Ludmil, and my sister Denitza for their love, encouragement and support during the years. Thank you!

Chavdar Bachvarov
Delft, The Netherlands
July 4, 2005
1.1 Problem description

A project called "Gatewayed Services" was launched within Philips Research that targets the possibilities of providing various Internet services to customers in one system. Within this project the concept of Gatewayed Communities has been developed. These are web communities of people who have something in common: share the same goal, interests, or relationship. Gatewayed Communities are deployed on the so-called Community System (CS). The CS is intended to be a service provider available on the Internet that offers services to the community members, such as peer-to-peer text, audio and video communication, asset sharing and set-up of other multi-party applications.

Due to the growing amount of personal digital content, sharing of assets with friends, relatives or colleagues on a community basis is of particular interest as a feature in the CS. The content, however, is private and needs to be protected from undesired usage. At the same time community members must be sure that the content is genuine and comes from the source they expect.

1.2 Objectives

The main objective of this project is to investigate the security architectures for sharing personal assets in communities deployed on the CS. The architecture of the community system as a whole and other potential features offered via such a community service, are outside of the scope of this project. The assumption is that the CS supports the management of communities, members and community assets. A prototype of such a CS was developed earlier.

The basic approach for sharing assets in a community is through copying the assets to a community owned asset pool. The (copy of the) asset is given to the CS, which has the capacity for storing the assets, becomes owner of the asset and can manage the access control (granting access to the shared assets only to community members). Another approach is that the assets remain with their original owner (in his/her home network) and only a reference to the asset is shared in the community. This project will focus on the latter approach and research security architectures that allow members of the community to access the shared community assets in the homes of the community members. Security includes protection of the communication channels as well as authentication and authorization of the access to the personal content shared within the communities.
1.3 Project approach

For the realization, initially the focus is on a solution based on a simple HTTP-server as service provider in the home that manages the shared assets and does the access control. The task will be to investigate possible security solutions for realizing the community access to the assets, and design and realize a prototype that demonstrates it.

The second phase investigates the impact on the security architecture if the assets are not managed on a separate server but are distributed over several devices in the home controlled by e.g. UPnP with an access control mechanism based on UPnP Security. In that context the concept of UPnP communities is developed and presented. A UPnP community is a distributed UPnP network, which unites the local networks of people that are members of a community. Within this network members share some of their services with other members, using the UPnP mechanisms for discovery, execution and protection of resources.

1.4 Report outline

The objectives of the project are explained in Chapter 2 together with a number of user scenarios, based on which the high-level system requirements are derived. Chapter 3 presents a number of technologies that have been selected to be used within the project. Chapter 4 motivates and describes the architecture of the Asset Community System, based on the the high-level requirements. The realization of this architecture is described in Chapter 5. Chapter 6 describes the UPnP communities - distributed UPnP networks, which unites the local networks of the members in a community. The related work is presented in Chapter 7. Chapter 8 concludes the report.
In this chapter we present the existing Community System (CS) and the features which the demonstrators of the system support. Next, we elaborate the goals of the project in a number of new features of the CS. Based on these new features we describe scenarios, which we think are necessary for the CS to operate. Finally, from these scenarios we derive requirements for the system architecture.

2.1 The existing Community System

Gatewayed Communities are intended to be a service provider available on the Internet that organizes common communication means for users such as peer-to-peer text, audio and video communication, asset sharing and set-up of other multi-party applications. The generic requirements for this architecture are:

- leverage existing Internet services;
- have simple and reliable user interaction to Internet services;
- guarantee privacy of consumers and their data;
- interoperability with PCs.

Figure 2.1: Conceptual architecture of Community Gateway

Figure 2.1 depicts the conceptual architecture of the Community Gateway. Basically, it works as a broker between services available online and devices connected to it. It
is suppose to attract and advertise different services to the connected customers. An example of a service the Community Gateway could offer is an Instant Messaging, broadly used for many years.

The Community Gateway has structures in which it keeps information about the community and the users registered on the gateway. The model is based on two directories: a people directory and a community directory, shown in Figure 2.2. The first one contains user accounts with information, such as address, private buddy list, assets, visibility for communities, etc. The latter describes the communities. Each community has a name, members, administrators, and other roles (e.g. asset writers: members that can add assets to the community). Further it holds a tree with the shared assets in the community, sub communities, presence information, etc. There are three types of communities - open, closed and private. Communities of the first type allow read access for its static information (e.g. name, assets, member info, etc.); however its dynamic information (e.g. presence) is not accessible from outside. Closed communities show only their name, description and administrators. This is the necessary information for membership application. The last type does not publish any information about itself, not even its name. The model applies the access control for non-members recursively to the sub-communities. Thus only open and closed sub-communities are visible to non-members. Closed and private communities allow read/write access only to members. The administrator(s) of the community grant roles to members; some of these roles may have (limited) write access for the community directory.

To prove these ideas, Philips Research developed two demonstrators [24, 34]. The first one offers members presence, peer-to-peer audio and video communication as well as asset sharing services. It consists of Community Gateway and two client applications - Device and Management Client. The Gateway accommodates the Data storage of the communities (OpenLDAP\(^1\)) and a presence server (Jabberd\(^2\)). The presence server is connected to popular Instant messaging systems, providing these services to the users. The Device Client is an application for the ordinary community members. It can be

---

\(^1\)http://www.openldap.org/
\(^2\)http://www.jabber.org
used for peer-to-peer text/audio/video communication and it connects to the presence server via XMPP protocol [28]. The Management Client is developed as a tool for the community administrators. With this tool they update the communities, add members, etc. It directly connects to the Data storage server using the LDAP protocol [35].

Figure 2.3: Implementation view of the first Gatewayed Communities demonstrator

The second demonstrator converges the communications with the CS to a Web application server (Tomcat 4). It supports Single Sign On (SSO) based on Liberty Alliance technology [1]. With SSO community members have access to service providers that are in alliance with the CS (e.g. other community systems). The demonstrator also offers a simple asset sharing service. Users can publish assets that are available on the Internet. They do that by inserting a name, URL and short description of the asset. If a user wants to view this asset he/she clicks on the link and is directed to the asset storage. The asset storage is outside the CS and could be a server that hosts e.g. pictures of the publisher, an interesting web site, etc.

2.2 New features introduced

The objectives of our project are to investigate technologies and secure architectures that provide secure access to the CS resources as well as to allow distributed secure sharing of asset within communities. The shared assets are supposed to reside at users’ personal device (at users’ homes). The security in asset sharing is limited to access and does not include control over the transmitted assets. The latter is targeted by the DRM technology and is outside the scope of this project. The distribution of assets and secure access features are further elaborated in this section.

2.2.1 Distributed asset storages

One approach to share personal assets is to upload them to a centralized asset pool at the CS. This is a popular method applied by many Internet-service providers, which deliver personal web-space services (e.g. Yahoo). The growing amount of personal assets, however, increases the demands from such services with respect to capacity, availability,
CHAPTER 2. COMMUNITY SYSTEM SPECIFICATION

performance, etc. Thus a major improvement of the CS would be to allow the community members to share their assets, while the assets reside at local Internet-enabled devices at users home. Instead of content, the users publish references to that content. This feature serves two goals. First of all, the feature reduces the demands from the centralized asset store. In addition to that, the users do not upload the asset content to a remote server, which avoids duplication of content as well as Internet traffic and publication time. Second, the assets remain (more or less) physically controlled by the users. They can always add/remove assets and/or unplug the device in case they detect malicious behavior toward their content (or device).

2.2.2 Content protection

Storing assets on the users’ device, however, is not enough to protect the data from unauthorized access. For example, information such as the location of the storage device, reference to an asset, etc., should not be given to outsiders or unauthorized members of a community. We distinguish two types of privacy - privacy in general (e.g. protecting data on operation system level) and privacy in the context of the Community System. The latter is of interest to us, because if the users share their assets within a community, they “trust” the CS that it shows these assets (and the asset existence) only to authorized members. In that respect we identify three types of threats concerning the assets:

1. eavesdropping (sniffing communication channel)

2. manipulation of communication channel (injecting and/or filtering malicious messages)

3. asset stealing (unauthorized access to the asset content)

The first type targets the prevention of channel sniffing from third parties as well as identification of the communication parties (to prevent the former). It is important to authenticate who you are talking to, since you may give out content to someone who is not part of the community. The conversation should be unreadable for third parties, namely encrypted, so that nobody can listen and copy the digital assets, while they are transferred.

The second type concerns changing or injecting messages in the communication channels (message integrity), as well as check of the genuity of the message source (message authentication). For example an attacker injects messages with different source addresses in order the asset to be sent to those addresses.

The third type of threat is related to unauthorized access to asset content. Only people who are granted access to the asset should be able to obtain the content.

In the Asset service implemented currently, the protection of the asset content is not part of the CS. To access an asset the user could face the protection mechanisms of the third parties that store the asset. The content protection feature eliminates this problem and allows users to access assets within the CS without necessity to login again.
2.3 Basic Scenarios

The above features describe the goals of the project. In order to motivate them, we developed a number of user scenarios which describe the steps the users and/or their devices need to take in order to use the new features. The technical scenarios are presented in Appendix D.

To make the story more realistic we consider the example of a community called "Photographer", hosted at our Community System (CS). This community unites several people in their photography passion. Within this community members share their experience, interesting stories, as well as the pictures they took. John and Steve are two members of that community.

2.3.1 Device Registration

Recently John bought a new Internet-enabled device, capable of storing pictures. Using that device, he wants to share pictures with the other members of the "Photographer" community. However, the device is unknown for the CS, thus John needs to register it there. Initially, John installs on the storage device an application, which can communicate with the CS (alternatively the application could have been installed in advance). Then he connects the device to the Internet. Via this community application John logs into the CS and registers the storage device at the CS, with the device address and identity. Once the procedure is completed successfully the device is ready to be used within the CS.

2.3.2 Asset Publication

Now that John has a registered device in the CS, he would like to share a couple of pictures with the other members of "Photographer". First he starts the community application and logs into the CS. Once logged in he can view the information in his community. The community contains a folder called "Assets". This is the root directory, in which community members share their assets. Using the community application John creates a sub-folder in the asset tree. Then he selects the pictures he wants to share and puts them into this directory. Finally, John publishes the references to the assets at the Community System.

In addition, we may consider the following variation on the scenario. John is traveling back home by train after his excursion in the mountains. He has the storage device with him. John uploads the pictures from his camera to the device. He opens the community application and since the device is offline John does a "local log" in. Next he creates a folder in the community asset tree, selects the pictures and puts them into this directory. Finally, when John is home and the device is online, he synchronizes the changes with the community tree.

2.3.3 Access to Assets

While browsing within the community tree with his community application, Steve spots the new pictures published by John. He wants to have a look at some of them. He picks
one and clicks on it in order to open it. This action starts a process, which downloads
the picture and opens it in a corresponding application. This process is transparent for
Steve (see Appendix D.3 for technical details). His experience is just ”click and view”.
Now Steve can see the pictures and e.g. print them.

2.3.4 Device Presence

Mobile storage devices are not always online from the same location. For example, con-
sider the variant on the ”Asset Publication” scenario. By default the device is registered
with an address from John’s home network. After the trip in the mountains John decides
to visit his parents, who live in a city near by. His family has an Internet connection, so
John can publish the pictures of his vacation right away. He plugs the device into the
network and it automatically contacts the Community System to announce its online
status. The CS identifies the device and acknowledges its presence. In addition, if the
CS detects that the device has new address it updates the contact information. Now the
device is again available to handle requests for assets.

2.4 System Requirements

So far we discussed several new features of the Asset Community System. Based
on these features we are going to derive the high-level requirements of the system
architecture. The requirements are separated in three sections: General, Distributed
storing of assets and Security and privacy requirements.

General requirements:

1. Reusing available technologies - a major requirement for the architecture is to reuse
   already available technologies.

2. The Asset Community System should be easy to use - as less as possible user
   actions necessary to operate in the system. For example, the user should know
   nothing more than how to install and work with a standard user application and
   a browser for Internet.

3. Flexible client side application - users should be able to access the Community
   System from any location that has Internet access (e.g. Home, Internet Cafes,
   Hotels, etc.).

Distributed storing of assets:

4. Publish assets stored on personal device - users should be able to share assets
   stored on Internet-enabled device(s) of their possession. The device(s) could be
   either mobile (MP3 player, PDA, etc.) or non-mobile (e.g. PC, Internet-enabled
   TV, etc.).

5. The storage device should be registered at the CS - the CS should be able to
   identify each device that shares assets. Thus users should be able to register their
   storage devices at the CS.
6. A device may store assets shared in more than one community.

7. More than one user may publish assets stored on the same device.

8. The asset location should be transparent to the user - the user should not be aware of the asset location.

9. The Community system should be notified when a device is online in order to update device connection information - often a mobile device migrates from one network to another, and this should not disrupt the asset sharing service.

**Security and privacy:**

10. The storage device and the CS should establish a trust relation - a registered device and the CS should exchange identity information, so that they can authenticate each other.

11. Single Sign On access to assets - once logged in the system, users should be able to securely access assets on different storage devices without the necessity to enter their credentials again.

12. The CS manages the access control to assets - the CS is the authority that asserts the remote storage device to grant a user access to a certain asset.

13. The exchange of messages between the involved parties in the CS should be confidential - the communication channels should be encrypted.

14. The content of the assets published in the CS should be protected - only authorized members are granted access.
As mentioned in Section 2.4 the use of existing technologies and Off-the-shelf components (COTS) is a major goal of our project. This chapter presents a number of technologies, which are (potentially) useful for the CS architecture. The list is not exhaustive, but focuses on the key technologies.

We start with LDAP as an option for the community repository. Next, we describe Kerberos and Public Key Infrastructure (PKI) as security architecture. Web Services are presented as a mechanism for remote and platform independent communication between software applications. Finally, Universal Plug and Play (UPnP) and UPnP Security are explained as a peer-to-peer mechanism for discovery, use and protection of services within local networks.

3.1 Lightweight Directory Access Protocol (LDAP)

The Lightweight Directory Access Protocol (LDAP) [18, 35] is a lightweight client/server protocol for accessing a directory service. It was initially used as a front-end to X.500[4], but can also be used with stand-alone directory servers. LDAP is optimized for read access and, thus, is usually deployed in environments where data is read very often but rarely changed. This property makes it an excellent candidate for the Community System repository, since community properties are more frequently read than updated. LDAP is available both in free (e.g. OpenLDAP [22]) and commercial (e.g. eDirectory [20]) implementations.

In order to import and export directory information to or from LDAP-based directory servers, or to describe a set of changes which are to be applied to a directory, the LDAP Data Interchange Format (LDIF) [13] is used. LDIF files store their information in an object-oriented hierarchy. The information model of LDAP is based on entries. An entry is a group of attributes-value pairs. The attribute has a type, which is usually a mnemonic string, such as cn for common name, or ou for organization unit. As an example consider cn with value "My Community", or ou attribute with value "People Directory". The most important attribute is the Distinguished Name (dn). Each dn must be unique, because it allows finding the corresponding entry in the LDAP structure. In LDAP, directory entries are arranged in a hierarchical tree-like structure. Traditionally, this structure reflects the geographic and/or organizational boundaries. Entries representing countries appear at the top of the tree. Below them are entries representing states (regions) and national organizations. Below them might be entries representing organizational units, people, printers, documents, or just about anything else one can think of. Thus a descriptive name would look like: "cn=Steve, ou=People,
3.2 Kerberos

Kerberos [30] was created at Massachusetts Institute of Technology (MIT) as a solution to network security problems. The name Kerberos originates from Greek mythology: it is the three-headed dog that guarded the entrance to Hades. MIT offers a free implementation of this protocol on its homepage\(^1\).

Kerberos is a network authentication protocol that is designed to provide strong authentication for client/server applications by using secret key cryptography.

Kerberos is based on the Needham-Schroeder authentication protocol (see Appendix C).

\[\text{Figure 3.1: Kerberos Overview}\]

The participants in a Kerberos system are depicted in Figure 3.1. The client is software that "speaks" the Kerberos protocol. When a user intends to get a certain service from a service provider the following actions take place:

1. The client sends a Ticket Granting Ticket (TGT) request to the Key Distribution Center (KDC).

2. The KDC returns a TGT with a corresponding session key encrypted with a key generated from the user’s password. That allows only this client to use this TGT. All user passwords are stored in a central database of the KDC. After that, the user at the client computer is prompted for his password. His password is used to compute the key that is able to decrypt the session key received from the KDC. If

\(^1\text{http://web.mit.edu/kerberos/www/}\)
the user entered the correct password he obtains the TGT and its associated key. Usually, this entire process already happens when the user logs into the client.

3. With the TGT, the user can ask for a service from a service provider. The client requests the Ticket Granting Service (TGS) for a service ticket by sending the TGT and a service request to the TGS.

4. The TGS looks in its master database for an entry for the client and the requested service. If the entry exists, the TGS issues and returns a ticket for this service.

5. The client sends this service ticket to the service provider that verifies the ticket using its own service key.

6. If the ticket is valid, the service provider now knows the identity of the user at the client computer and is able to provide the service. The precondition is that every participant is registered with a username and password at KDC.

3.3 PKI

Trust among people (and organizations) has always been an important issue. Usually face-to-face contact, photos and written signatures are used to identify people. However, this is not enough in the anonymous world of Internet.

Public Key Infrastructure (PKI) is an attempt to transfer the trust in the physical world to the anonymous Internet. It is based on asymmetric key cryptography with a public and private key pair. Each subject shares its public key with others, while keeping the private key at a safe place. The following example demonstrates the use of the key pair:

User A wants to send confidential data to user B and she wants to be sure that only B can read the message. A has B’s public key and uses it to encrypt the message. As long as B possesses the private key, he can decrypt the message. Reversely, in order A to prove to her recipients that a message comes from her, A signs the message with her private key. Consequently, every recipient, who has A’s public key, can verify the message and be sure that she is the sender.

The public key is part of a digital certificate of a subject (person/organization). The certificate also contains information such as common name, address, issuer, encryption algorithms etc. The certificate is issued and signed by a Certificate Authority (CA). The CA takes care of the link between the physical subject and the digital certificate. The signing of subjects’ certificate basically states: ”If you trust this CA you may trust this certificate and the subject behind it.” It is interesting the mention that the common name in the certificate of a web server is the name with which clients connect to it. Usually this is the DNS name (e.g. the common name in the digital certificate of the ABN AMRO web server is www.abnamro.nl). Alternatively, it could be the IP address of the web server.

Furthermore, the CA is responsible for the registration, revocation, suspension, and other manipulation with certificates (see Appendix G for PKI definitions and Appendix B for x509 certificates).
3.3.1 SSL/TLS

The Secure Sockets Layer Protocol (SSL, [10]) is a protocol developed by Netscape, which is designed to provide privacy between two communicating applications (a client and a server) by using public key cryptography. Additionally, the protocol is designed to authenticate the server and, optionally, the client. SSL requires a reliable transport protocol (e.g. TCP) for data transmission and reception. An advantage of the SSL protocol is that it is application protocol independent. An application level protocol (e.g. HTTP, FTP, Telnet, etc.) can operate on top of the SSL protocol transparently, without any changes to the implementation. The SSL protocol negotiates an encryption algorithm and a session key as well as authenticates a server before the application protocol transmits or receives its first byte of data. All application protocol data is encrypted before transmission, ensuring privacy. The connection provided by the SSL protocol has three main properties:

1. The connection is private. All messages are encrypted using secret key cryptography (e.g. DES, RC4, etc.) with a session key that is defined at the beginning with an initial handshake.

2. The identities can be authenticated using public key cryptography (e.g. RSA, DSS, etc.). The server endpoint of the conversation is always authenticated, while the client endpoint authentication is optional.

3. The connection is reliable. The protocol includes a message integrity check using a Message Authentication Code (MAC) ensuring that package alteration between client and server is detected. The MAC is calculated using a secure one-way hash functions (e.g. SHA, MD5, etc.).

Transport Layer Security (TLS) [6] is the latest enhancement of SSL. The TLS protocol is based on the SSL 3.0 protocol specification as published by Netscape. The differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate. The major changes are cryptographically stronger MAC computation, larger padding (up to 256 instead of 63 bytes) and some protocol cleanup (e.g. ignoring unknown record types, improved alert messages, etc.).

The OpenSSL project [32] offers an Open Source implementation of the SSL/TLS protocols. The project is based on the excellent SSLeay library developed by Eric A. Young and Tim J. Hudson and is managed by a worldwide community of volunteers.

3.3.2 HTTPS

Hypertext Transfer Protocol over Secure Sockets Layer (HTTPS) is used in order to provide cryptographically secure web connections. All common browsers support HTTPS, thus, no additional installation is needed on client side. HTTPS connections are often used for submitting private information (e.g. credit-card numbers, personal details, etc.). On the server side there are SSL/TLS modules for most of the common web-servers (e.g. Apache, Microsoft Internet Information Server, etc.).
There is a project called mod_ssl\(^2\) that provides strong cryptography for the Apache web-server via the SSL/TLS protocols by means of the OpenSSL toolkit described above.

### 3.3.3 IPSec

IPsec stands for Internet Protocol Security. The IPsec protocols were developed by the Internet Engineering Task Force (IETF)\(^3\) and will be a part of IPv6. They are also widely being implemented and used for IPv4. IPsec uses strong cryptography in order to provide authentication, privacy and data integrity at the protocol level of the network protocol stack. These services allow building of secure tunnels through non-trusted public networks. It can be used together with PKI. Everything passing through the non-trusted network is encrypted by the IPsec gateway machine on one side and decrypted by the gateway machine on the other end of the tunnel. The result is a Virtual Private Network (VPN). IPsec uses three different protocols:

1. **AH** Authentication Header provides a packet-level authentication service.
2. **ESP** Encapsulating Security Payload provides encryption plus authentication.
3. **IKE** Internet Key Exchange negotiates connection parameters, including keys, for the other two protocols.

### 3.4 Web Services

Web Services (WS) [27, 29] are a technology that provides a standard way of interaction between different software applications that are deployed on various platforms and/or frameworks. The large use of XML [2] increases the interoperability and extensibility of WS. XML allows Web Services to be combined in a loosely coupled way in order to perform complex operations. Apart from XML, Web Services are based on Simple Object Access Protocol (SOAP), Universal Description, Discovery and Integration (UDDI) and Web Services Description Language (WSDL).

**SOAP** is a protocol specification that defines a uniform way of passing XML-encoded data. It also defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol.

**UDDI** provides a mechanism for clients to dynamically find other web services. Using a UDDI interface, clients can dynamically connect to services provided by external partners. A UDDI registry is similar to a CORBA trader, or it can be thought of as a DNS service for business applications. A UDDI registry has two kinds of clients: businesses that want to publish a service (and its usage interfaces), and clients who want to obtain services of a certain kind and bind programmatically to them.

**WSDL** provides a way for service providers to describe the basic format of web service requests over different protocols or encodings. WSDL is used to describe what a web service can do, where it resides, and how to invoke it. UDDI registries describe

\(^2\)http://www.modssl.org/  
\(^3\)http://www.ietf.org/html.charters/ipsec-charter.html
numerous aspects of web services, including the binding details of the service. WSDL fits into the subset of a UDDI service description.

3.5 UPnP

For the second part of the project we investigated Universal Plug and Play (UPnP). UPnP enhances peer-to-peer network connectivity for personal computers, wireless devices, and other intelligent appliances, in a distributed, open networking architecture. UPnP uses existing standard protocols, such as TCP/IP, Hypertext Transfer Protocol (HTTP), and Extensible Markup Language (XML) to seamlessly connect networked devices and to manage data transfer among connected devices.

UPnP provides an architectural framework for creating self-configuring, self-describing devices and services. Networks managed by UPnP require no configuration by users or network administrators because UPnP supports automatic discovery. UPnP enables a device to dynamically join a network, obtain an IP address, and convey its capabilities on request. Control points can use the UPnP application programming interface (API) to learn about the presence and capabilities of devices that are registered on the network. A device can leave a network smoothly and automatically when it is no longer in use.

UPnP uses no device drivers, as we know them. The protocol is media-independent and can be used on any operating system (OS). UPnP enables control over a device user interface through the browser and offers programmatic control to applications. UPnP enables developers to write their own user interfaces for devices, forgoing the vendor-provided interface.

3.6 UPnP Security

UPnP Security \[9, 7, 8\] is an extension of UPnP protocol that is intended to provide strict, but at the same time flexible control over resources within a UPnP network. The main functional components in UPnP Security are the Control Point, the Security Console and the (Secure) Device (see \[7, 8, 9\]). Each component is equipped with a globally unique public/private key pair, used for authentication and for encryption. Each participant has a Secure ID, equal to the SHA1 of its public key that uniquely identifies it.

A Control Point can execute actions on a Device. The Security Console manages the access permissions for the Control Points on the Devices. The Device checks the access permissions per action in its own administration. The latter consists of an Owners list, Access Control List (ACL), and Certificates in the Certificate Cache. If the Control Point has additional certificates it sends them to the Device Certificate Cache before invoking the action. A Security Console owns multiple Devices, and Devices can be owned by one or more Security Consoles. The Device owner fully controls the Device, bypassing any ACL and Certificates. It also inserts Access Control Entries in the Device’s ACL that define “who” can “when” do “what”:

- “who” represent an ACL subject, which is either anyone, or an identifier (typically a Control Point), or a Named Group of Control Points. The latter consists of a
Text Name and Key issuer (usually a Security Console).

- "what" covers the Access Permissions and Delegation. The former says what action is allowed to the ACL subject. The action could be, for example, browse, play, etc. Additionally, "what" says if the ACL subject is allowed to delegate its permission to other subjects.

- "when" defines the validity period in which the ACL subject is granted the Access Permissions. It indicates the validity not before date/time and/or not after date/time.

Apart from the Access Control Entries, the access control is managed by certificates. These certificates are signed by Security Consoles and issued to Control Points, which later present them to the Devices. There are two types of certificates - Authorization and Named Certificates. The first one is actually an Access Control List entry signed by a Security Console. This mechanism is supported by UPnP Security, since not all Devices have sufficient memory to store ACLs. A Named Certificate states that a Control Point belongs to a certain Named Group of Control Points. The combination between Named certificates and Named Groups reduces the number of Access Control List entries. The UPnP Communities architectures, described later in this report, relies on the latter mechanism.

Appendix E provides more information about these certificates as well as the interfaces of the UPnP Security participants.
This chapter describes the asset aspects of the architecture of the Community System architecture, which we call Asset Community System architecture. The chapter starts with a rationale of the selected security architecture. Next, the directory model of the Community System (CS) repository is presented. Additionally, the conceptual view of the system is described. Finally, the components of the architecture and the deployment view are presented.

4.1 Security architecture selection

In the previous chapter we presented two security architectures (PKI and Kerberos), which are of potential interest to our project. We analyzed both of them in order to understand which one complies better to the system requirements. The main focus was on secure, authorized and at the same time flexible access to community resources.

We started studying Kerberos for several reasons. First of all Kerberos provides strong authentication, confidentiality and data integrity. Additionally it uses strong cryptography to protect the secrets and symmetric keys within the system. The password of the client is required once per session and access is provided to all service providers registered in the system. Last but not least, the model of Kerberos is very similar to our vision of the distribution of the assets (see 4.2). The CS stores the user and device credentials, just like the Key Distribution Center (KDC) does. Additionally, user storage devices could be registered as service providers at the KDC.

However, we found difficulties utilizing authorization within Kerberos. Even though a Kerberos ticket contains an ”authorization-data” field \[23\], we found its usage inadequate for our purposes. The intended usage is that the client (not the KDC) puts some authorization data in that field. This data is e.g. a certificate, or a password, which the user possess permanently, or acquires from the CS on a regular basis. The authorization data is also application specific (e.g. \textit{User X is member of community C and D}). Additionally, we assume that the users in the system are dynamically changing their roles, access rights etc. Thus using Kerberos would mean that clients should initially fetch authorization data from the CS (see requirement 9), via or outside Kerberos, and then launch another (this time) Kerberos ”access to service provider” request, which includes the assertion. We think this communication is redundant. Additionally, Kerberos demands from the client and the service provider to support protocol specific modules. However, not many off-the-shelf components support these modules, which would increase the integration costs.

Thus we decided to investigate the Public Key Infrastructure (PKI). In PKI each
device, as well as the CS, is equipped with public/private key pairs. Such a key pair enables devices to sign/verify and encrypt/decrypt messages, in a way that third parties cannot easily decipher them. The infrastructure allows the CS to compile messages, dedicated to the service providers, which authorize users to access assets. The messages could be generated dynamically (including the very current status of the users), and even more - the users are not aware of the message content, which we think is good for the security properties of the system (the less one knows the better). The CS is able to support this functionality. The storage devices should also be able to support it since they should have enough resources to store and deliver assets. In addition, we assume that the devices are capable to generate their key pairs and send their public key to the CS. The latter is necessary for the CS to create a server certificate for the devices and affiliate them to the CS trusted environment. Exchanging asymmetric (public) keys between the devices and the CS is better from security point of view than exchanging symmetric keys.

Finally, PKI collaborates with a number of secure protocols (see Chapter 3.3). One of them, HTTPS with SSL/TLS, is supported by every browser\(^1\).

---

4.2 Directory Model of the Community System

The directory model of the Community System is an instance of the Gatewayed Community conceptual model described in Section 2.1 and [33]. However, we made two extensions of this model according to the system requirements (section 2.4).

First, we added a device directory to the model (see Figure 4.1), which is required for the registration of the storage devices. The device directory holds the description of the devices registered in the system. Each device has an identity (hash of the public

---

\(^1\) Actually this is one of our targets for client application
key), descriptive name (dn), IP address, owner information, etc. The device directory is not open for general browsing, because of privacy reasons. However, devices are visible within the communities of their owners.

Second, the model of the asset in the community tree is extended with a unique identifier and a reference to the device (in the device directory) that stores the asset. The former is generated during the asset publication to uniquely identify the asset in the system. The reference to the device is used to get at run-time the up-to-date address of the device.

![Conceptual view of Asset Community System](image)

Figure 4.2: Conceptual view of Asset Community System

### 4.3 Conceptual view

The conceptual view of the CS with distributed and secure asset sharing is presented in Figure 4.2. The picture shows the three main types of participants in the system: Community System (CS), Asset Browser and Asset Writer\(^2\). The CS is the main party. It stores the information about the communities as well as people’s and devices information. The Asset Browser (AB) is a user who can browse the community and access assets via a Community Client (CC). The CC is an application running on a device of the user. The Asset Writer (AW) is a user who is able to publish assets in a community\(^3\). They use an Asset Provider (AP) to publish assets. The AP is an application installed on an Internet-enabled storage device of the AW. It consists of Asset Publishing Component (APC) and Asset Server Component (ASC). With the first component the Asset Writers publish assets in their communities (Section 2.3.2). Additionally, they use it to register their APs in the CS and exchange identification information (Section 2.3.1). The second component works as a service at the device and provides access to the published assets (Section 2.3.3).

\(^2\)Later we use *user(s)* when referring to Asset Browser and Asset Writer.

\(^3\)One user can have both roles.
It also announces its online presence to the CS (Section 2.3.4). The medium between the parties is Internet so they should communicate via encrypted channels to ensure security.

### 4.4 Security model

The goal of the Security model is to establish mutual trust between communicating parties as well as data integrity. The model explains the basis on which the participants exchange information in a trusted way, but not the type of information itself. The model relies on PKI and additional assertions, communicated between the participants. As explained earlier each party has a public/private key pair that identifies it in the system. This holds only for the CS and the APs (their ASC components), but not for the users. They are registered at CS with username and password. The CS also plays the role of a Certificate Authority for the registered devices. It issues them server certificates using the mechanism mentioned in Section 3.3. The CS architecture is open for mobile devices, which (often) change their residence network, respectively their IP addresses. This makes it difficult to assign a DNS name for each device, because by the time the Internet DNS tables are updated, the device could change its address again. Thus, we decided to set the current IP address of the devices as a common name in the certificates. However this obliges the devices to change their server certificate together with their IP address.

The model relies on three assumptions and a number of assertions. The first assumption is that the community members are registered in the system, by an authorized person (community administrator(s)). Second, the Asset Provider has been registered at the CS by a registered user. Finally, in order to share assets in the community, the user of the AP has been given Asset Writer (AW) role by the community administrator(s).

![Figure 4.3: Security model of Community System](image)

---

4With IPv6 this problem will disappear
4.4. SECURITY MODEL

Figure 4.3 presents the security model of the system at run-time. The assertions between the parties are as follows:

1. User to the CS: The username-password combination represent the same person, who registered earlier at the CS - this assertion is valid, except for the case when the identity of the user has been stolen (or voluntarily disclosed to someone else). For that reason we should make sure that the username/password pair is not transmitted as plain text.

2. The CS to User and AP: This is the same Community System, as the one to which the user and the AP have been registered - the assertion can be ensured by equipping the CS with a server certificate, which is presented to the user (respectively the application used by the user) and AP (the ASC component) each time they communicate with the CS. Actually PKI complies with this solution, since the secure protocols rely on such certificates. Additionally the server certificate of the CS should be proliferated in broader context (medium), so that users and APs can verify it from independent sources.

3. The CS to Asset Browser: Assets, which users may find in their communities, are published by an Asset Writer and the content is stored on a device, trusted by the CS - assets are published in a community only by authorized members. A member should have an Asset Writer role in that community in order to publish assets. This level of protection is introduced to control asset publishing\(^5\).

4. AP to the CS: The signed messages are sent by the same device that possess the private key corresponding to a public key CS trusts - the AP has been registered with its public key in the CS and has proven to the CS that it possess the corresponding private key.

5. The Asset Browser to AP: This user is allowed to access this asset - this assertion is used in case the user request an asset from an AP. The user obtains a proof from the CS, which allows him/her to access the asset. The proof is automatically communicated to the AP. The structure of the proof is initially agreed between the AP and the CS, so the AP is able to verify that the proof comes from the correct CS, as well as to validate it. In our model the proof is valid for certain period of time (the demonstrator uses 30 seconds).

6. AP to Asset Browser: This AP is the same AP specified by the CS you are registered with - this assertion can be met in two steps. First, the AP should be equipped with a server certificate, which is presented each time a user contacts the AP and which states that indeed this is the correct AP. Second, the certificate of AP should be signed by the CS (CS as Certificate Authority), so that the user to be sure that the AP is trusted within the system.

Distribution of secrets

\(^5\)More about roles in community one can find in [34]
From the security model described above we can derive what secrets each party needs to store.

The CC, used for browsing the communities and obtaining assets, deals with the smallest number of secrets, since it knows only the certificate of the CS (which is actually public). The CC deals with the username and password of the Asset Browser, but does not permanently store them. They are inserted at run-time and are valid for a session.

The AP possesses relatively small number of secrets as well. It stores its public/private key pair and the certificate of the CS. The public key is accompanied with a server certificate signed by the CS.

The participant with the largest number of secrets is the CS. It stores the username/password pairs of the users. In addition it keeps the public keys of all devices, and its public/private key pair associated with a CS server certificate.

Figure 4.4: Security architecture of Asset Community System

4.5 Asset Community System Architecture

Figure 4.4 shows the security architecture of the Asset Community System. It is composed of a community repository as a back end and an application server as a front end. They communicate via LDAP(S), since the former is an LDAP server. The Asset Provider (AP) device (bottom left corner) hosts the AP components. The latter use SOAP over HTTPS to communicate with the CS. The user interaction device, on which the Community Client (CC) runs, communicates with the CS and the AP device via HTTPS. These communications are expressed in Figure 4.4 with solid lines. The CC in our architecture is a conventional browser, thus we are not going to discuss it further.

The dotted arrow in Figure 4.4 expresses that the community repository has (indirect) references to the AP devices, respectively the assets stored there. The references are indirect in the sense that links behind the assets point to the Community Gateway Application Server (CGAS). Thus the Asset Browsers request the assets from the CGAS, which redirects them to the actual location of the assets.

4.5.1 Community Gateway Application Server

The front end of the CS architecture is the Community Gateway Application Server (CGAS). It accommodates the Community System Application (CSA), which provides
interfaces for the CCs and the APs to access the repository of the system. The architecture of the CSA has 3-tiers: Presentation, CS Logic, and Back end, which are depicted in Figure 4.5. The solid line expresses the LDAP(S) communication between the Back end tier and the community repository. The dotted lines depict the dependencies between the tiers.

The Presentation tier has two interfaces, one is for the HTML clients (CCs) and the other for SOAP clients (APs). The architecture, however, is open for other interfaces (e.g. for WML clients).

As the name suggests, the CS Logic tier accommodates the logic of the application. Amongst others, it implements the following services:

- **User Login** - logs the user in and out of the system.
- **Asset Publication** - gets a list of assets, parses it and inserts it in the repository.
- **Asset Requests handling** - receives request for an asset, generates an assertion and attaches it to the address of the AP, which stores the asset. The result is a URL, which is returned to the CC. Each assertion expires after a limited amount of time. This prevents reply attacks on the AP. In case the connection to an AP is slow and the assertion arrives too late, the request will be cancelled. However, a new assertion could be generated, since they are issued each time an asset is requested.
- **Device Registration** - obtains a certificate request from a device. Extracts the public key from the request and stores it. Generates and returns a server certificate, which device has to store.
- **Device Presence** - obtains a message from a device, identify it and updates its address (if necessary).

---

6The URL is in the style of https://AP_address:port/asset_application/asset_handler?assertion=value
The first four functions are invoked by a user, thus the request contains the user credentials. The last one is called by the Asset Server Component (ASC) of the AP and is signed with the AP credentials (private key).

The **Back end** facades the data model of the community repository. It implements the communication with the community repository as well as defines a number of interfaces for the basic operations such as insert, update and obtain data. These interfaces decouple the application logic from the community repository implementation, which makes the CSA architecture open for different types of data storage servers (i.e. LDAP server or RDBMS).

Finally each tier of the architecture uses a number of libraries, common configuration properties, and data structures summarized as Utilities. Apart from the standard libraries, Utilities include libraries for encryption/decryption, manipulation of server certificates, etc. Additionally, the CSA has its own data model representing the entities (i.e. devices, assets, users) in the system.

![Figure 4.6: Asset Provider Architecture (modules view)](image)

### 4.5.2 Asset Provider Applications

The Asset Provider application runs on the asset storage device. It enables the device to participate in the Asset Community System. The application implements the scenarios described in Chapter 2. The architecture of the AP application is presented on Figure 4.6. It consist of two components (1) **Asset Publisher Component** (APC) - a client application, which provides user interface of the Asset Provider, and (2) **Asset Server Component** (ASC) - a server that handles the asset requests. The functionalities of the AP application is distributed between these two applications.

The APC implements the registration of AP and asset publications, functions which require user interaction. In order to provide offline publication it also caches the Community view of the user. The application communicates securely with the CS, thus it uses the device security cache, where the device stores its key pair, as well as the trusted certificates. Additionally, the application reads from the file system the location of the assets. Finally the published assets are inserted in the Published Asset List (PAL). This XML file maps the representation of the asset in the community on the local file system, providing asset location transparency. In the PAL, assets are presented with their
identifier, which makes it possible to synchronize the PAL with the Community view, in order to prevent redundancy in the PAL. In the device registration procedure the application obtains signed server certificate from the CS, which is stored in the device security cache.

The ASC is responsible to serve asset requests and to announce its online presence to the CS. The component is an HTTP server with a community specific application deployed on it. When the ASC receives a request for an asset, it first verifies the source and validates the request. Then the application extracts the asset identifier and finds the corresponding local path in the PAL. Finally it returns the asset content.

Both applications run as independent processes. The Asset Server Component is envisioned as a service running at the device, while the Asset Publisher Component is envisioned as a stand-alone application. The touch points of the two processes are the read/write operations in the PAL as well as the common security cache. In addition, the Asset Server Component should refresh the server certificate after registration or after the certificate is updated.

Figure 4.7: Community System Information Flow

4.5.3 Information Flow

Figure 4.7 summarizes the information flow between the participants in the Asset Community system. Each arrow shows the direction of the flow as well as the type of the information and the credentials that signs the information. Some of the assertions are exchanged in a session (e.g. when the user is logged in the CS), while other information is send accompanied with the credentials of the sender (e.g. device presence).

The users and the CS authenticate each other with their credentials - username/password for the users (/AB/AB, /AW/AW); and the server certificate for the CS (/CS/CS). The CS delivers the community view to the Asset Browser (AB) “signed”
with CS private key ([community view]CS). Signed means that the CS delivers the community view in an authenticated HTTPS session. When a AB requests an asset from the CS, the latter sends a ticket signed with its private key ([ticket]CS). The ticket is redirected to the Asset Provider (AP) in an asset request from the Asset Browser ([ticket]CS). The AP authenticates before the AB with its certificate, which is signed by the CS ([AP]CS). The AP also proves that it holds the private key corresponding to the public key in the certificate ([AP]AP). Finally, the AP returns the asset in the opened session ([asset]AP).

Via the Asset Publisher Component, the Asset Writer registers the Asset Provider ([AP]AW) and publishes assets ([assets]AW). On registration, the CS returns the server certificate of the AP signed with CS private key ([AP]CS). The Asset Server Component updates its presence with messages signed with the private key of the AP ([presence]AP).

![Figure 4.8: Community System deployment view](image)

### 4.6 Deployment view

Figure 4.8 presents the deployment view of the system. The Community System (CS) application and the Asset Server Component (ASC) of the Asset Provider (AP) are deployed on web application servers. The Community Client (CC) is a browser, which communicates with the CS and the AP via HTTPS. The Asset Publisher Component of the AP connects with the CS via SOAP over HTTPS. It is a stand-alone applications.
5 Asset Community System Architecture realization

The previous Chapter explained the architecture of the Asset Community System. To prove the architecture is realisable we implemented the described components. This Chapter starts with details about the design of the Community System Application and the Asset Provider. Next, the organization of the code is presented. Finally, the Chapter provides instructions how to build, configure and install the components.

5.1 Asset Community System Application Design

The Asset Community System consists of a J2EE [21] web application deployed on a Jakarta Tomcat server [14] and a community repository stored in a Novell eDirectory LDAP server [20]. The web application uses Novell LDAP libraries to extract information from the repository and offers HTML and SOAP interfaces to its clients. The HTML interface is implemented with Java Server Pages and Java Servlets [16], while Apache Axis [15] provides the SOAP interface.

5.1.1 Information Model

The information model of the CS Application (CSA) complies with the repository model discussed in Section 4.2. All entities in the LDAP repository have unique distinguishable names. They belong to three major directories - People, Communities and Devices - which belong to the ”dc=Philips, dc=com” root. The CSA for secure asset sharing, however, targets on three of the entities - Person, Asset and Device.

The Person represents the properties of a user. CSA uses this entity to authenticate and authorize the user access to the community repository, thus the interesting properties are the username, password and role in the system. The UserDetails class represents this entity.

The Asset encapsulates meta-data for assets, such as name, asset identifier, publisher identifier, community path, URL, storage device identifier and description. The community path is the descriptive name (dn) of the asset in the repository\(^1\). The URL points to the asset within the community system, which provides transparency of the asset location. The properties of this entity are represented in AssetDetails class.

Finally the Device has a descriptive name (dn), security identifier, IP address, labeledUri and friendly description. It is interesting to mention that the device identifier in the Asset entity of the Asset directory is the dn of the device in the system. The dn

\(^1\)For example cn=MyAsset,cn=assets,ou=MyCommunity,ou=Communities,dc=Philips,dc=com
is derived from the device security identifier\(^2\). The latter is supplied by the devices on registration and is actually the hash of the public key of the device\(^3\). The \textit{DeviceDetails} class represents this entity in the application.

### 5.1.2 Modules view

This section describes the three tiers and the modules inside them. Figure 5.1 zooms in on the modules of the CSA tiers, explained in Section 4.5.1. It shows the dependencies inside and outside the tiers.

![Figure 5.1: Community System Application Architecture (modules view)](image)

#### 5.1.2.1 Back End tier

The \textbf{Back End} tier represents the community repository in the CS application. The tier uses \textit{Abstract Factory} pattern \cite{11} to decouple the CS application from the repository type. This makes the system flexible when migrating to different types of storages. It provides four interfaces to the CS Logic tier - \textit{AssetManager}, \textit{DeviceManager}, \textit{UserManager} and \textit{SystemManager}. The first three offer get/set methods for the corresponding entities. \textit{SystemManager} retrieves information about the Community System from the repository such as the public/private key pair of the system. The implementation of the Back End in the CS demonstrator works with an eDirectory LDAP server and depends on Novell libraries to connect to the repository.

\(^2\)In our implementation the device \textit{dn} looks like "\textit{cn=device1234567890, ou=Devices, dc=Philips, dc=com}"$, where the digits after \textit{device} are the integer hash code of the device security identifier. The latter, which is the hash of the device public key, cannot be entered in the LDAP server as \textit{cn}, because of the symbols it contains.

\(^3\)We use SHA1 hash function.
5.1. ASSET COMMUNITY SYSTEM APPLICATION DESIGN

5.1.2.2 CS Logic tier

The CS Logic tier implements the logic of the services offered by the CS. These are asset services (publication and access), device services (registration and presence) and user login services. Considering that classification, they are distributed in three modules Asset Service, Device Service and Login Service. The three of them use the Back End interfaces in order to load and store the data from/into the repository. Each module is implemented in a class with the same name (see Figure 5.2).

![Figure 5.2: Community System Application Class Diagram](image)

The Asset Service offers two services: Asset Publication and Asset to Access. The Asset Publication is called from Asset Writer users, thus the request is accompanied with user credentials. The input for the service is a list of entries with Asset meta-data. The publishing service inserts this information in the repository together with the dn of the storage device and a generated URL to the asset.

The Asset to Access service is called by the Asset Browser. It receives as input parameters the community path to the asset and details for the asset requester. The service extracts from the repository the Asset meta-data together with the public key of the storage device. Then a ticket is generated, encrypted and attached to the URL of the device. The modified URL is returned to the asset requester.

The generated ticket consists of the Community System name, the asset identifier and the time of the request. It is first encrypted with the RSA algorithm using the CS private key as encryption key and then again with the RSA algorithm this time using the device public key as encryption key (so only the device can read the ticket). The output of the first encryption authenticates the source of the ticket, whereas the second encryption ensures that only the device that holds the private key can understand the data in the ticket. Finally, the ticket is encoded with Base64 and UTF-8 algorithms in order to make it transportable as a parameter in an URL. The Asset Service uses the Bouncy Castle security provider implementation for the encryption operations.

---

4 The URL is similar to https://qas.philips.com/community_application/asset_request_handler?path_to_asset=cn=MyAsset,cn=assets,ou=MyCommunity,ou=Communities,dc=Philips,dc=com

5 http://www.bouncycastle.org/
The **Device Service** has two main services: *Device Registration* and *Devices Presence*. The *Device Registration* registers new devices in the repository. This service is invoked by registered users, thus the request contains the user credentials. The service receives a X.509 certificate request with the device details (see Appendix B), together with the device secret identifier (hash of the public key) and description of the device. The certificate request is generated by the device. The security identifier is signed with the device private key and the signature is also supplied as a parameter to the service. Initially the service verifies the input parameters. This is done in two steps. First the device security identifier is checked if it is equal to the SHA1 hash of the device public key (the latter is extracted from the certificate request). Next, the signature of the secure identifier is verified in order to make sure that the device holds the private key corresponding to that public key. Once data is authenticated, the service inserts the device meta-data in the repository. It also generates a new server certificate for the device. The issuer of this certificate is the subject in the CS server certificate. The device certificate is signed with the CS private key. Finally it is returned to the requester.

The *Device Presence* service is called by the Asset Server Component (ASC) to announce its online presence. The service gets the device secret identifier together with a signature of it (as in *Device Registration*). The function verifies the signature and extracts the IP address of the source from the HTTP message context. This address is compared with the one in the repository and if they are different, the function updates the repository with the new address. Finally it returns notification to the device if the address has been changed.

In addition, the module provides `updateCertificate` function for devices to update their server certificates in case they changed their address. The function follows the certificate generation procedure explained in the *Device Registration*. The output is a server certificate, which is sent to the device.

To accomplish its services, the **Device Service** module uses the Bouncy Castle libraries. They are useful for signing the device certificates as a Certificate Authority.

Finally the CS application has a **LoginService** module, which authenticates users. The module provides interfaces to the HTML and SOAP front end for logging users in the system. The `login` function obtains the username and the password of the user and return `UserDetails` object.

### 5.1.2.3 Presentation tier

The **Presentation** tier of the CS demonstrator for secure asset sharing consists of HTML and SOAP modules. The HTML interface offers to clients the *Access to Asset* service (*AssetRequestHandler* servlet). The SOAP interface is used for the *Asset Publication*, *Device Registration* and *Device Presence* services. Since SOAP requests are stateless, each of them is accomplished with the username and password of the requester. The exception is *Device Presence* service, which is called by the ASC and is authenticate by the signature of the input parameter. The SOAP front end invokes the **Login Service** to authenticate the users requesting the service. Once the requester is successfully au-
5.2 ASSET PROVIDER

The Asset Provider consists of two components - the **Asset Publisher Component** (APC) and the **Asset Server Component** (ASC) (See also Figure 4.6). The first one offers an Asset Writers interface for publication of assets to the their communities. The **Asset Server Component** provides and protects the access to the published asset content. The two components do not have direct communication, however they exchange information via Published Asset List (PAL). Additionally they share the same secret cache that stores the public/private key pair of the device, as well as the Community System certificate.

5.2.1 Asset Publisher Component

This is a Java application built with the the Java Swing Graphic User Interface (GUI). It offers publishing functions to the users, as well as Asset Provider registration functionality. The application has three main modules - **AssetSelector, AssetPublisher** and **RegistrationManager**. Additionally it has a **ConnectionManager** module. Figure 5.3 depicts the class diagram of the component.

![Asset Publisher Component class diagram](image)

The **AssetSelector** obtains the asset’s **community path** and **local path** from the GUI (inserted by the user) and compiles meta-data for it. The meta-data additionally consists of **publisher username**, **asset name**, **asset identifier**, **action** on the asset and **description** (remarks). The **asset identifier** is generated for newly added assets and is
the result of the SHA1 hash function applied over the concatenated string of ",publisher name + community path + device id + local path". The action field states if the asset is added or removed from the community. The meta-data entry is inserted in the Offline Published Asset List (oPAL), which is part of the local tree cache. Future refinement on this module could include publishing of multiple assets, directories or local directory (sub)tree.

The AssetPublisher module publishes the selected assets in the community. It obtains from the AssetSelector a list of selected assets and sends it to the CS. The CS responds with a list of the accepted asset identifiers. These assets entries are removed from the oPAL and inserted in the Published Asset List (PAL), making them available in the system. The asset entries in the PAL consists of asset identifier and local path.

An alternative design for the AssetPublisher would be to read the oPAL directly, in stead of invoking the AssetSelector. However, since both modules are in the same application we decided that in order to improve the read/write synchronization it is better if the AssetSelector works exclusively on the oPAL.

The CS could reject some of the asset entries, because e.g. user do not have sufficient rights, or for any reason the asset path in the community does not exist, etc. In that case the system returns reason for rejection.

The RegistrationManager module registers the device in the CS. It generates an X509 certificate request with the device key pair. The request is sent to the CS together with the device identifier and signature of that identifier. The CS responds with server certificate, which is inserted in the device security cache. Finally the Asset Server Component (ASC) is restarted in order to load the new certificate. In our implementation, the registration finishes with configuration of the ASC to use the device secret cache. However, in a real application the ASC could use default configurations. The module use the Bouncy Castle libraries to generate the certificate request.

Finally, the application has a ConnectionManager, which establishes the connections with the CS. It is aware of the syntax and the parameters necessary to make remote procedure calls on the SOAP services offered by the CS. Thus one could consider it as a repository for the services offered by the CS.

5.2.2 Asset Server Component

The Asset Server Component in our demonstrator is a Tomcat web application server \[14\], on which a community application has been deployed. Tomcat is capable to establish HTTPS connections with the asset requester. The server extracts its server certificate from the device secret cache, which in our demonstrator is a Java key store file\[7\].

The Asset Server Component has two main modules: the AssetProviderHandler and

---

\[^6^]{We do not consider time, because that would increase redundancy in the system. The same asset published by the same user from the same device in the same community several times would end up in different assets for the system.}

\[^7^]{http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html}
the PALManager. The PALManager reads the Published Asset List (PAL) and extracts
the local path of a requested asset. The PAL is an XML file, with (asset identifier,
asset local path) entries (see Appendix F). Figure 5.4 presents the class diagram of the
component.

![Asset Server Component class diagram](image)

Figure 5.4: Asset Server Component class diagram

The AssetProviderHandler serves the requests for the assets. It extracts the ticket
from the URL and decodes it in the reverse sequence as encoded. The module applies
UTF-8 and Base64 decoding. Then decrypts the ticket with the Asset Provider’s private
key and again with the CS’s public key. The successful decryption assures the AP that
the ticket comes from a genuine CS. Apart from the verification of the authority that
issued the ticket, the AssetProviderHandler calls a TicketValidator module to check the
recency of the ticket. If the time stamp of the ticket and the current time at the AP
differ more than 30 seconds (in our demonstrator) the ticket is rejected.

5.3 Code organization

The project structure of the applications is presented in Figure 5.5. communityApp
is the root directory of the project. The deliverables for the three applications (Asset
Community System, Asset Publisher Component, Asset Server Component) are in this
project folder. src contains the Java source files of the application. It has common
com/philips/ folder in which the applications have their specific directories. build is the
output directory for the compiled classes. dd contains library, configuration and any
other files, added to the applications during packaging. Finally jar stores the ready-to-
install files.

Each application use a common util package. It contains the data model classes (As-
ssetDetails, UserDetails, DeviceDetails and TicketDetails) as well as ConfigurationMan-
ager class, which reads the system configuration properties. A SecurityManager class,
which works with Java key store, is available in security subpackage. The folder also
contains CertificateUtil class for X.509 certificate requests generation and server certificates signing. Finally the crypto subpackage contains CipherUtil class with sign/verify, encrypt/decrypt, encode/decode functions. The last two classes use the Bouncy Castle libraries.

5.3.1 Asset Community System Application

The Community System application of our demonstrator consists of three packages, one for each tier (see Figure 5.6). The repository directory contains the Back End tier interfaces and their LDAP implementations. The service package includes the CS Logic tier classes. The HTML component is in servlet folder. The application also includes the gws packages from the existing CS demonstrator. The web/gws folder contains HTML interface files. Finally the above described util package is added.
5.3.2 Asset Provider

5.3.2.1 Asset Publisher Component

Figure 5.7 shows the Asset Publisher Component code structure. gui directory contains the Graphic User Interface classes of the application. The rest of the modules are distributed in corresponding folders.

5.3.2.2 Asset Server Component

Figure 5.8 shows the code structure of the Asset Container application. asset package contains the PALManager module. security directory holds the TicketValidator. The AssetProviderHandler is in servlet folder. The application also includes the util package.

5.4 Project Build

The description of the build process assumes that Apache Ant [17] has been installed. The three applications are compiled, packaged and deployed using Ant. The root di-
CHAPTER 5. ASSET COMMUNITY SYSTEM ARCHITECTURE REALIZATION

The directory contains `build.properties` and `build.xml` files which contain the commands for building the application. The first file contains deployment and library paths necessary for the commands in the second file. All commands presented below are started from the same directory as `build.xml` (the project root directory).

### 5.4.1 Community System Application

The Community System Application is compiled by the following command executed from the application root directory:

```
ant compile-community, compile-gws
```

To package the web application the following command is executed again from the root directory:

```
ant package-gws-web
```

Figure 5.9 shows the structure of the output `gws.war` file. The root directory contains the web files (jsp, html). The `lib` directory contains library files used by the application. These are Apache Axis, Novell LDAP, XML parser and Bouncy Castle jar files. They are obtained from the `dd` directory. Finally the configuration files (`Configurations.properties` and `Init.properties`) of the application are in `classes` folder.

![gws.war file structure](image)

Figure 5.9: gws.war file structure

### 5.4.2 Asset Provider

The following commands compile and package the Asset Publisher Component:

```
ant compile-assetpublisher

ant package-assetpublisher
```
5.5. CONFIGURATION

The output, `assetpublisher.jar` file, is created in `jar` folder of the project.

The Asset Server Component is created via the following commands:

```
ant compile-assetserver

ant package-assetserver-web
```

The output, `assetserver.war` file, is created in `jar` folder of the project.

The installation `assetprovider.zip` file is prepared by running the following command:

```
ant package-assetprovider
```

This command depends on the `package-assetserver-web` and `package-assetpublisher` commands. The output zip file has two sub-directories - `assetserver` and `community`. The first one contains the `assetserver.war`. The other one stores the Asset Publishing Component, which includes `assetpublisher.jar`, configuration files (`Configurations.properties`, `Init.properties`, `assetProvider.keystore` and `assetpublisher.bat`) from `communityApp/dd/assetPublisher` and library files (from `communityApp/dd`). The `assetProvider.keystore` contains the CS server certificate.

5.5 Configuration

The Community System has a `Configuration.properties` file with system environment properties. The file is in classes folder of `gws.war`. It includes the path to the `communityServerKeyStore.keystore` (accompanied with the password and alias for the community key and certificate), the path to the temporary folder (where the received server certificate requests are stored), the type of the repository, the URL of the asset handler module, etc. (see Appendix F). Finally, the Tomcat server needs to be configured to use the community secret store. The configuration consists of adding the following `Connector` tag in the `Service` tag of the Tomcat `server.xml`:

```xml
<!-- Define a SSL Coyote HTTP/1.1 Connector on port 8443 -->
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
  port="8443" minProcessors="5" maxProcessors="75"
  enableLookups="false"
  acceptCount="100"
  debug="1" scheme="https" secure="true"
  useURIValidationHack="false"
  disableUploadTimeout="true">
  <Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
    keystoreFile="c:/jakarta-tomcat/conf/communityServerKeyStore.keystore"
    keystorePass="community"
    clientAuth="false" protocol="TLS" />
</Connector>
```
This configuration of Tomcat implies that the security cache file is in \texttt{c:/jakarta-tomcat/conf/communityServerKeyStore.keystore} and the password for the cache is \texttt{community}.

The Asset Provider (AP) components share the same \texttt{Configuration.properties} file. It consists of the paths to the shared key storage (accompanied with the necessary passwords and aliases), the Asset Server Component, PAL and oPAL XML files. Additionally it stores the time skew for the ticket (see Appendix F). The configuration file resides in the deployment directory of the Asset Publisher Component.

The configuration file of AP applications is made for Windows platform, however if the paths are changed the programs can run e.g. under Linux.

\section*{5.6 Installation}

The installation of the CS application requires \texttt{gws.war} file to be copied to the \texttt{webapps} directory of Tomcat server. Alternatively the following command would deploy the application in case the tomcat deployment dir in \texttt{build.properties} is correct.

\begin{verbatim}
     ant deploy-gws
\end{verbatim}

The installation of CS application requires \texttt{gws.config.xml} and \texttt{model2.xml} to be copied from \texttt{communityApp/dd/gws} to \texttt{C:/LDAP_gws_configuration}. Make sure that the \texttt{communityServerKeyStore.keystore} is in \texttt{c:/jakarta-tomcat/conf/}.

The CS application uses Apache AXIS web services. Each time the application is deployed the services need to be registered at the Community Gateway Application Server (CGAS). The description of the services is in \texttt{deploy-community.wsdd}. The following command, started from the project root directory, sends the service descriptions to the CGAS:

\begin{verbatim}
     communityAXISdeployer deploy-community.wsdd
\end{verbatim}

The \texttt{communityAXISdeployer.bat} assumes that the CGAS runs at localhost and that the AXIS lib files are in \texttt{c:/axis/lib} directory at the same machine where the bat file runs. In case one (or both) parameter is different they should be updated in \texttt{communityAXISdeployer.bat}.

The AP software is installed on the asset provider device. It requires installation of Tomcat (v4 or v5). The application specific files are in \texttt{assetprovider.zip}. Unzipping \texttt{assetprovider.zip} in \texttt{c:/} creates a directory \texttt{c:/community/} and \texttt{c:/assetserver/}. Then copy \texttt{assetserver.war} from \texttt{c:/assetserver/} to \texttt{TOMCAT_ROOT/webapps} (Tomcat 5.x should automatically deploy the application without restart, Tomcat 4.x requires restart).

The Asset Publisher Component is in \texttt{c:/community/}. \texttt{c:/community/lib/} contains the following files: \texttt{soap.jar, axis.jar, commons-discovery.jar, commons-logging.jar, jaxrpc.jar, saaj.jar, log4j-1.2.8.jar, bcprov-jdk14-128.jar, mail.jar, activation.jar}

To start the Asset Publisher Component, run \texttt{assetpublisher.bat}. To run the Asset Server Component start the Tomcat server.
This Chapter investigates the possibilities of secure sharing of resources in the style of services instead of content. The research is in the context of UPnP networks, where devices’ resources are shared as services. UPnP Security is the security mechanism we consider in these networks. Additionally the concept of UPnP Communities is described, as a platform for sharing services between personal UPnP networks.

6.1 UPnP communities - from content to services

Community System (CS) and UPnP communities are an area where people with UPnP networks can share services and resources with other people on a community basis. A UPnP community is a distributed UPnP network, which unites the private UPnP networks of the community members. In contrast with content sharing, where references to the content are uploaded to a central server, the service descriptions are distributed among the personal networks. They are discovered with the means of UPnP discovery protocols [12]. In contrast with Asset Community System, where the CS centrally stores asset information and is responsible to announce the asset presence, the distribution of services in UPnP Communities and their dynamic discovery eliminates the problem of keeping the service presence up-to-date. However, users need to query every network in the community in order to find the services they might be interested in.

The community participants need not necessary know each other personally (even though the most common scenario is that the communities are established by friends, family, etc). As long as they belong to one community, the participants can access resources shared by members of that community.

Additionally, each UPnP community is a secure sub-domain in the CS area of trust. Each participant relies on the CS as a central authority. The CS builds trust between members, by the means of assertions. These contain information about the status and/or permissions, which the members (respectively their devices) have in the community.

6.2 Basic Scenarios

As a beginning, we consider several scenarios, which in our opinion are important for the UPnP communities. We motivate them with the example of the ”Photographer” community discussed earlier (see Section 2.3). In this new set up, the community members are equipped with UPnP networks in their homes. John and Steve are still members of the community. The access control mechanisms is based on UPnP Security [7] (see also
Section 3.6). We assume that in these networks reside UPnP security-aware devices as well as that a Home Security Console (HSC) is available.

6.2.1 Registration of UPnP network at CS

Recently John equipped his home with a number of UPnP security-aware devices such as media server, TV, printer, and other storage devices, organizing a UPnP network. Additionally, John uses a gateway device (also known as Home Gateway, HGW) as a connection point of the home network with the ”outside world”. Apart from the conventional methods, John controls this equipment with a number of control points, installed for example on his PC, wireless PDA, etc. To protect the access to the equipment John uses the UPnP security mechanism [8]. However, at this moment John is still not able to share and/or consume services in the ”Photographer” community, because his network is not registered at the CS.

Similar to the Device Registration scenario (Section 2.3.1) John registers his home network in the Community System. The home network is represented by its home gateway, as a device connecting it to the Internet. Initially John installs the community application on the home gateway (alternatively the applications could have been already installed). John makes sure that the gateway device is connected to the Internet. Via the user interface of the community application John logs into the CS and registers the device. The registration procedure includes exchange of addresses and identities between the CS and the device.

Now the network of John is part of the CS trusted environment. In addition, the network is automatically assigned to the communities, in which John participates.

6.2.2 Discovery of UPnP Services

Once John has registered his network, he is ready to consume the services shared in the community. The first step is to discover them. Via the community application John connects to the CS and obtains a list with the addresses and identities of the community members’ networks. With this information he (automatically) connects to these networks in order to build the distributed UPnP community\(^1\). Once the connections are established, the connecting module exchange service descriptions with the other networks. The descriptions are automatically distributed (using UPnP mechanisms) to the control points and devices within John’s network. At the end of this procedure John is able to see the services shared in the community and is ready to use them.

6.2.3 Publication of UPnP services

In the mean time Steve has shared several services from his networks to the ”Photographer” community. The publication is based on the UPnP Group access control mechanism mentioned in Section 3.6 (see also [9, 7, 8]). First, the community application downloads from the CS description of the communities to which Steve belongs. Next these descriptions are communicated to Steve’s Home Security Console (HSC). Among

\(^1\)Establishment and support of all connections between the networks of community members is resource consuming; alternatives are discussed later in Section 6.5
them is the "Photographer" community. Steve sees "Photographer" as a UPnP Group at his HSC. Using the standard UPnP access permission mechanism, he allows the community and its members to access browse and play actions at his media storage service. This process accomplishes the publication of services with a community. Now each control point that has a group certificate for membership in "Photographer" community, is able to access the Steve’s media server.

6.2.4 Access to UPnP Services

Now let us return to John, who just discovered the services offered by Steve. Using his control points (CPs), he is about to approach Steve’s media server. However, John’s CPs need certain access permissions. Those are issued by the CS, as a governor of the "Photographer" community. The community application sends to the CS a permission request list, which consists of description of the control points (with their SecurityIDs) in John’s network. The CS, based on John’s membership in "Photographer", issues permission to each control point in the list to access the resources of the community. Then the CS returns the permissions to the community application, which distributes them in John’s network. This process is automatic and transparent for John. The internal distribution is done within UPnP, as explained in [9].

6.3 System Requirements

The above described scenarios give us a basis to derive the high-level requirements for the Community System and UPnP communities. The requirements go beyond the scenarios considering other aspects of the system (e.g. multiple users in one network). We divide the requirements in three groups - General, Security and Additional requirements.

General requirements:

1. One user may participate in more than one community.
2. More than one user may use one UPnP network.
3. Within a UPnP community, members share services instead of content.
4. The location of the services provider device is transparent within the community.
5. The service descriptions are distributed among the local networks, in contrast to storing them on a central server.
6. When two (or more) networks connect to each other within a community, they exchange the descriptions of the services available in each network.
7. The CS provides information about the community members’ networks.
8. Users access services with their control points.
9. Users should be able to consume services offered in networks of other community members

**Security requirements:**

10. The networks of the members should be able to establish a secure (encrypted) connection.
11. The networks of the members should be able to authenticate each other.
12. If access to a service is granted to a community, all members should be able to consume it.
13. Users should be able to control the exposure of their UPnP networks and services in the community (against discovery and access).
14. In order to be protected by the CS mechanism a device should be UPnP Security aware.

**Additional requirements:**

15. The system should allow finer grain access control than community groups (e.g. access to certain community member(s), or control point(s)).
16. A control points should be authorized by the CS in order to access a service shared in the CS.
17. It should be possible to revoke the authorization of the control points.

### 6.4 Community System model updated

Based on the above information we can design a model for the UPnP communities. This model is an extension of the Gatewayed Communities model described in [33] and similar to the one discussed in Section 4.2. In [33] there are two directories, community directory and people directory. The first one stores information about the communities: name, members, administrators, roles, etc. The second directory stores information about the people registered at the communities. It contains user’s identity, address information, and other personal information. We extend this model with a third directory that holds information about the UPnP networks of the users. This information includes: friendly name of the network, IP address, identification information, network owner, as well as other data. Additionally, the entities in the people directory are extended with a field that stores a list of UPnP network(s), from which the user wants to share services. This list contains references to the network directory.

In this model we distinguish four classes: Community system, Community, Person and UPnP network. Figure 6.2 shows the associations between them. The communities, people and UPnP networks are registered at the Community System. People are
6.5 Conceptual View

In this section we describe the conceptual view of the UPnP communities architecture. The goal is to unite the local UPnP networks of community members into one distributed

members of zero or more communities. A community may also have no members, if the administrator is an outsider for that community. Each UPnP network is associated with at least one person (usually the one that registered the network), and at the same time one person can be associated with many UPnP network. By default, the community membership of a person makes the UPnP networks, associated with him, part of that community.

Figure 6.1: Conceptual Model of UPnP Communities

Figure 6.2: Associations in UPnP community model
CHAPTER 6. UPNP COMMUNITIES

UPnP network. Within this network members share and consume services as if the UPnP Community network is homogeneous. To achieve the goal we need to take two steps:

1. Connect the local UPnP networks
2. Distribute access control within the UPnP community network.

The connection of the UPnP networks is peer-to-peer. Solutions regarding the latter are already available [5].

For the second step we use as a starting point the UPnP Security mechanisms. As explained in Section 3.6 each secure UPnP network has (at least) one security console, through which the users distribute access permissions to the devices in their UPnP network. These permissions usually state that certain control point can execute certain action on a device. Additionally, at the security console users can group control points and give access permissions to a group instead of to an individual control point. This coarser granularity is easier to manage and can be applied for community groups.

Figure 6.3 shows a possible application of that concept in the CS: the Community System (CS) runs a security console (SC-CS), in which different community groups are defined. From UPnP Security [8] point of view, users "define" at that console the community groups, which are allowed to access their devices. Additionally, each control point (CP) gets a certificate, which states a relation to a certain community group (in this document we use also assertions to describe the UPnP community groups and UPnP certificates, issued by the CS). At a later stage a control point approaches a devices as explained in Section 3.6. All these assertions are distributed to the private UPnP networks according to the relations in our model for the CS.

![Diagram of UPnP communities]

Figure 6.3: Pre-mature conceptual view of UPnP communities

This concept, however, does not consider two important factors. First, in order to distribute access permissions to a device, a security console should be an owner of that device. In this model this means that the security console of CS is owner of the local
user devices. Second, UPnP communities are composed of a number of private UPnP networks, which are entirely controlled by their owners/users (we assume by means of UPnP Security). In that case there should be a "home" security console, with which users distribute access permissions, since this security console "owns" the home devices (Section 3.6).

An alternative solution of the problem is to make CS security console co-owner of the device. However, we should consider the fact that users should be in full control of their devices and resources. Thus, in Figure 6.4 we propose an elaborated conceptual view.

The Community System (CS) is given the role of authority, which issues and distributes UPnP assertions. Once the assertions reaches the "entrance" of the private UPnP network it is user’s responsibility to accept them and allow internal distribution or not.

The distributed UPnP community network is supposed to be a homogeneous UPnP environment. Consider the following example: three community members Alice, Bob and Eve belong to one community and have their networks connected in one UPnP network. Each of them discover the services offered by the others. Now consider that Bob has shared a play service at his media server with lots of songs and Eve has shared a service on a render device. In a homogeneous UPnP network it should be possible for Alice to play music from Bob’s music server at Eve’s rendering device. In practice each local UPnP network of the three members is connected peer-to-peer, to the networks of the other members. This leads to a fully connected graph of UPnP networks. Suppose a community has N members, this makes N-1 connections opened at each user’s network, altogether N*(N-1)/2 connections. For large N the support of N-1 connections is significant, which is not desired.

Figure 6.4: UPnP communities conceptual view
A possible optimization is when users connect their networks within the community on demand (e.g. Alice connects to Bob in order to check the services he has published). This optimization would reduce the run-time load with regard to connectivity, but community members still need to connect to all other networks in the community in order to get all published services.

Another aspect of the UPnP communities is the overlapping of the communities’ domains. It should not be possible for a control point to connect devices, which belong to different communities, even though the control point belongs to both communities. It occurs when a user belongs to two or more communities and exposes his/her services to the members of these communities, and also gets the services shared by these members.

Consider the above scenario with a modification, such that Alice and Bob belong to community A, while Alice and Eve belong to another community B. However, even though Alice discovers the services of Bob and Eve, she is not able to play the music of Bob on the Eve’s render. The first reason is that Eve and Bob belong to different communities, the Buddy lists of each of them does not contain the network of the other, making it impossible to establish trusted connection within the CS\(^2\). The second reason is that Eve’s rendering device does not have community assertions to approach Bob’s music server.

### 6.6 Security model

In our Security Model we consider the Community System (CS) as a trusted area that uses PKI as security architecture. Within that trusted area we have communities, users and UPnP networks collaborating. Each community defines its own secure sub-domain, in which users share services from their UPnP networks to other members.

The sharing is based on establishment of peer-to-peer connections between UPnP networks that belong to members of one community. From a security point of view, networks need to authenticate each other before they establish a connection. The authentication could be based on assertions from a trusted third party (as implemented in the Asset Community System), or on mutual knowledge about each other (e.g. a predefined Buddy list). The latter is a centrally stored Buddy list, which is distributed to every local network. The Buddy list is managed by the users’ communities and/or by the users themselves (private list). The architecture of the Community System (CS) is suitable for both approaches. In the first approach, both networks do not know about each other, but trust assertions issued by the CS. This is convenient in case the networks is not able to manage a list of friendly networks. In the second approach, the CS sends a list to each network with details of the networks to which they may connect, and which may contact them.

We choose the latter option for our Security model, since the general model of the CS, suggests that it is able to deliver such a list to the users. The list contains information about the communities the user belongs to, together with information about the networks of other community members, the buddies. So the users are able to connect to their buddy networks and establish a logical UPnP network, in which both networks notify

---

\(^2\)However, this does not exclude trusted connection outside the scope of the CS
each other about the services they offer.

Our Security Model also relies on UPnP security (Section 3.6). As mentioned before UPnP communities are secure domains, in which services shared in a community are available to all members. Therefore in our Security Model we set the authorization granularity to community groups. This means that users issue access permissions for a service to the whole group instead of to individual user (or control points). At the same time, the control points get certificates from the CS, which state that the control point belong to certain community group. With these certificates the control points approach the devices to ask for services.

6.7 System Architecture

This section presents the architecture of the UPnP communities. It describes the components necessary to build the system.

Figure 6.5 shows the UPnP Community System architecture. The UPnP networks are represented by their Home Gateway (HGW) devices. The HGW is a router device, which amongst others, has a UPnP stack with which it communicates to the device inside the network. The HGW communicates with the Community Gateway via SOAP over HTTPS. Additionally the HGWs connect to each other with IPSec/GRE tunnels. The Generic Routing Encapsulation (GRE) protocol is broadly used for building Virtual Private Networks (VPN) because of its good routing mechanisms. Wrapping the GRE tunnel with IPSec secures the connection. As mentioned in Section 3.3 IPSec is compatible with the PKI security architecture of the CS. X.509 certificates are used for authentication.

Figure 6.5: UPnP Communities Architecture
6.7.1 Community Gateway

The architecture of the Community Gateway remains unchanged as explained in Chapter 4 (see Figure 4.5). The front end is the Community Gateway Application Server (CGAS), which supports HTTP(S) and SOAP interfaces. The CGAS communicates with the community repository (an LDAP server) via LDAP protocol.

The UPnP Communities, however, require deployment of additional services on the CGAS and update of the directory model of the community repository. The former are added as modules to the CS Logic tier of the CS application. The intention is that the local networks will approach these services via the SOAP interface. The directory model we discussed in Section 6.4. The new services at the CGAS are related to the scenarios described in Section 6.2. They are the following:

1. **Registration service** - the service registers the HGW device of the UPnP network. It reuses the device registration procedure presented before. The outcome is that the CS gets the public keys of the HGW and the HGW is equipped with a server certificate signed by the CS. It is important to mention that the HGW is registered only once by a person registered in the community system. This person is associated as a network owner.

2. **Association of UPnP networks to user’s network list** - the relation between local UPnP networks and users is many to many (requirements 2 and 3). The service allows users to extend the list of UPnP networks they use (e.g. home, vacation house, office, etc.). To prevent unauthorized association, this service is invoked from the targeted UPnP network by its owner and specifies the name of the user, to which the network is going to be associated. This user should be registered in the community system.

3. **Community view service** - the service provides a Community View to the UPnP network. The Community View consists of description of the communities, as well as a list of community members. The former is a UPnP Group description, which contains the CS security identifier and the name of the community. The CS here plays the role of a security console and each community is a group defined in the namespace of the CS. The community member list includes connection information about the members’ UPnP networks (such as address and public key/certificate).

   There are two alternatives for the invoker of this service. The first one is that the HGW gets the community view for each users associated to the network and distributes it inside the network. The other option is that the service is invoked by a user and returns his/her community view. It depends on the privacy policy of the Community System and within the local UPnP networks. However, the information model of the system is powerful and flexible enough to support both alternatives.

4. **Assertion service** - the service is invoked by the HGW to request assertions, necessary for the control points inside the network to operate within the communities. The HGW sends a list of security identifiers of the control points residing in the
home network. The service compiles UPnP Name Definition Certificates (see Appendix E), which state that a control point belongs to a community, as UPnP Group. The service issues certificates to each control point for each community the HGW is associated with.

These services are the basis that the CS should support in order to run the UPnP Communities.

### 6.7.2 UPnP network components

A local UPnP network plays the roles of service consumer and service provider within the UPnP Communities. Thus the local UPnP networks need components that would allow them to share services, but also to consume service. A UPnP network consists of UPnP Devices, Control Points and Security Console(s). They use UPnP protocols to discover and consume services. The UPnP Communities intend to preserve these mechanisms within the network, but at the same time make it open for the CS and other networks. A crucial role in achieving this goal is played by the HGW.

In the UPnP communities architecture the HGW is the device that connects the local network to the outside world by offering UPnP interfaces inside the local network. In our architecture it additionally has HTTPS and GRE/IPSec interfaces to the CS and other networks. The task of the HGW is to make the outside environment UPnP transparent for the devices inside. It means that a device or control point should contact devices outside the network the same way as they contact devices within the network. For that reason we concluded that the HGW is the device that is most suitable to accommodate the necessary CS components. In addition, we assume that the HGW has a security cache that stores certificates/public keys (e.g. of friendly networks).

Regarding the scenarios described in Section 6.2 we derived four main functionalities for the application we want to deploy at the HGW:

1. Registration of HGW at CS and association of users to the UPnP network.
2. Downloading the community view from the CS.
3. Obtaining assertions from the CS.

We distribute these functionalities in four components: Registration Manager, Community Manager, Assertion Manager and Tunnel Manager.

### 6.7.2.1 Registration Manager

The Registration Manager component participates in the device registration scenario (see Section 6.2.1 and Section 5.2.1). It allows the network owner to register its HGW device in the CS. The assumption is that the user is the administrator of the device.

---

3One can imagine many optimizations regarding the number of certificates issued by the CS. For example the input list could specify, the communities to which the control point applies for Named Group certificate.
in order to register it in the CS. As a result of the registration procedure explained earlier, the component receives a server certificate, which is inserted in the HGW device security cache. With this certificate, the device is identifiable and can establish secure communication.

Additionally, this component is able to associate users that use the network. The association is done by the network administrator. He/she sends a message to the CS, which message contains the username of the user that wants to be associated with this network.

The component uses SOAP calls to accomplish these operations. Within the network it has a UPnP interface and appears as a service of the HGW.

6.7.2.2 Community Manager

The Community Manager component downloads the community views of the users working in the local network. Amongst others, the community view consists of the users’ UPnP Community Group descriptions and a list of members of these communities. The information entry of each member is accompanied with a list of IP addresses - public key (certificate) of the UPnP networks they utilize. This list helps the local users to connect to their remote co-member networks. The public key (certificate) of each network is added as trusted to the security cache of the HGW. The component is responsible for keeping the trusted entries in the security cache up-to-date.

The component has a UPnP security console interface through which it offers the UPnP Community Groups to the network. The descriptions should reach the home security console, from which the user enables a community (respectively its members) to access local devices. Currently the exchange of Name Group descriptions between security consoles is not part of the UPnP Security Console interface. However, the descriptions could be added to the list returned by the \texttt{NameGroupList()} in security console interface (see [8]). Alternatively the UPnP Community Groups could arrive to the Community Manager component as certificates (as described in [7]). In that case they are available in the network by certificate.

Another aspect of Community Manager component is privacy. As mentioned before the community view could be obtained by the HGW for all users associated with it, or the component requires user credentials and delivers only private community views. From a security cache consistency point of view it is better to obtain all community views and filter them locally, and always have an up-to-date list of buddy networks.

6.7.2.3 Assertion Manager

The component works for the consumer network. It delivers assertions to the control points. The control points use these assertions to approach the devices that share services in a community. The Assertion Manager behaves as a UPnP Security console inside the network. Using UPnP it discovers the control points in the network. With the list of control points it approaches the CS Assertion service. The CS service generates and returns a list of Name Group Certificates for each control point, which the Assertion Manager stores and offers in the home network. At later stage the control points approach the Assertion Manager via UPnP and obtain the certificates issued to them.
6.7.2.4 Tunnel Manager

The component connects two UPnP networks in a transparent UPnP network, using a GRE/IPSec tunnel. The component resides at the HGW. The consumer network uses this component to connect to other networks. In the provider network the component accepts the connections based on the "Buddy list". Once the connection is established, the components in both networks listen for UPnP discovery messages and send them to the other network via the tunnel. To establish the connection, the component needs the address of the remote network. Since the connection is encrypted it uses the HGW security cache to consult if the certificate of the remote network can be trusted.

Additionally, the component supports four modes to filter the communication. The two extreme modes are to enclose or disclose every device. The third mode shows only UPnP Security devices. The last one is a customized view - the user decides which devices are announced in the remote network.

![Figure 6.6: UPnP deployment view](image)

6.8 Deployment view

The above described components provide all the information the users need to enable access to services from their networks within their communities. They also deliver the necessary certificates to the users’ control points, so that users can consume services in other networks. Figure 6.6 presents the deployment view of the UPnP Communities system. The HGW communicates with the CS via SOAP/HTTPS and with the other community networks via GRE/IPSec. Internally it has UPnP interface.
CHAPTER 6. UPNP COMMUNITIES

Figure 6.7 shows the distribution of the above described CS components on the HGW. The first three components are organized in one CS Bridge application deployed on the HGW. The CS Bridge application has a UPnP Security console interface to communicate with the devices and the control points within the network. It also has a SOAP/HTTPS interface to contact the CS. The Tunnel Manager is separated from the application because it can be used to establish connections outside the Community System. The Tunnel Manager refers to the HGW secure cache to identify the trusted remote networks. The CS Bridge is responsible to keep that list up-to-date.

![Figure 6.7: UPnP Communities network components (deployment view)](image)

Figure 6.7: UPnP Communities network components (deployment view)

![Figure 6.8: UPnP Communities Invocation view](image)

Figure 6.8: UPnP Communities Invocation view
Within the network, the CS Bridge application makes the information about the
UPnP community groups available to the home security console. The users grant access
permission to those community groups on the devices via the security console, which
communicates them to the devices. The control points fetch from the CS Bridge the
UPnP Named Group certificates generated from the CS. Additionally, the CS Bridge
offers its services to community user applications within the network which visualize
the community view and provide user functionality. Finally the community user appli-
cation invokes the Tunnel Manager in order to connect to other UPnP network in the
community. These relations are depicted in Figure 6.8.
7.1 SAML & Shibboleth

Security Assertion Mark-up Language (SAML) is an XML-based framework for creating and exchanging of security information between online partners [19]. The Security Services Technical Committee (SSTC) of the standards organization OASIS (the Organization for the Advancement of Structured Information Standards) develops SAML. The Shibboleth project of Internet2/MACE (Middleware Architecture Committee for Education) and IBM [3, 25] is an implementation of the "Destination-site first" scenario described in SAML. Shibboleth intends to provide authentication and authorization for web-based applications.

Shibboleth has the advantage that the user is able to control which information is transmitted to which resource. Authentication is done by the user’s home organization, authorization by the resource provider. Another advantage is that Shibboleth seems to be a scalable architecture that will be released as Open Source.

The disadvantages are that the Shibboleth is tightly coupled to web-based applications. In addition, the requirements for the resource providers are too high for storage devices. Furthermore the resource provider contacts the origin site (the Community System) two times per user resource request, which the Asset Community System architecture avoids.

7.2 VITELS

Virtual Internet and Telecommunications Laboratory of Switzerland (VITELS) [31] is a project that involves several universities and engineering schools in building a virtual laboratory where students can improve their skills in the realm of computer networks. The laboratory offers to the students online courses, developed in different universities, as well as access to resources of these courses.

The goal of the project is to provide secure and authorized access to the different resources offered in the system. For that purpose VITELS uses different secure protocols (e.g. IPSec, HTTPS, SSH) and PKI as security architectures. Each university supports its own student database, which students should be able to access in order to login to the system. The data storages are accessed via LDAP protocol.

VITELS is an example of heterogeneous system in which different types of entities form a secure environment. It is designed for interaction between organization servers (such as the Community System), but is not scalable in terms of dynamically and automatically adding and removing entities (devices) from the security environment. However
7.3 Groove Virtual Office

Groove Virtual Office\footnote{www.groove.net} is software package that allows distributed teams of people to work over the network as if they work in the same physical location. The application provides people with the flexibility to work on- and off-line, from their office, home, on the way, or at a customer site. The Groove architecture relies on the power of the desktop to put application capabilities directly in the hands of users. It uses servers as brokers to negotiate the connection between users who are not currently on the network, who are separated by firewalls, or who are connected in bandwidth-constrained locales.

The Groove application is installed on the user’s PC and does not rely on centralized servers. It stores application logic and data locally secured by encryption. The changes of the shared space are synchronized with the other team members.

The Groove infrastructure consists of three additional entities. Those are Groove Enterprise Relay Server, Groove Enterprise Data Bridge and Groove Enterprise Management Server. The first one is a store-and-forward back-up server which helps to synchronize data in special situations. It provides back-up of the shared space view to users which have been off-line for a long time, or cannot connect to other users for update. Additionally, people who cannot connect directly (e.g. because of a firewall) can use this server to communicate changes in the shared space. The Groove Enterprise Data Bridge is a seamless portal to external systems, such as Web services, database and enterprise applications, Customer Relation Management (CRM) systems, etc. It provides a service-oriented architecture, which offers the users to integrate external applications into the system. The Groove Enterprise Management Server deals with the user administration, license and security policies. The Enterprise Management Server acts as a central location for all Groove related management and administration activities such as PKI integration, identity disabling, data access control, etc.

The client side of the Groove architecture at this moment is heavy and requires installation on a PC (or a laptop). This implies that resources of the shared space are only available via this application, which is in contrast with the Community System, where people are able to view the community resources every where via a browser. Since the data in Groove is distributed, version control over the documents is an open issue. Recently the application is reported to support distribution of meta-data about the shared space instead of content. Additionally the encryption the data locally slows the performance of the application. In the Community System we locally encrypt and decrypt only meta-data instead of content. The content, however, is encrypted/decrypted during transfer in the secure channels.
8.1 Conclusions

This report presents the Asset Community System architecture, part of the ”Gatewayed Services” project launched by Philips Research. This system accommodates communities of people and their devices. The architecture was improved in two directions (1) sharing of distributed resources (both as content and as services) as well as (2) secure access to these resources within the Community System.

Selecting PKI as basis for the security architecture is considered to be a good choice. All necessary connections have been secured by already existing, well-known and robust, standard protocols. Additionally, all devices in the system that share services or assets are equipped with public/private key pairs, providing flexible authentication and authorization schemes. We also realized that currently, the most convenient method for user authentication on the web is via username and password. However, the architecture will not change significantly in case the user is equipped with a smart card or other means of digital identification and authentication.

A number of scenarios were developed in order to motivate the architecture decisions. These scenarios explain how users securely share and access content on remote devices within a community. A demonstrator was build to implement the asset sharing based scenarios and show the new features of the system.

In addition, the concept of UPnP communities was investigated, in which community members share services in their networks with other members of their communities. This concept is based on distributed UPnP networks that unite the local UPnP networks of community members. These communities use UPnP discovery protocols to find distributed service and UPnP Security to protect these services. The community members provide the access to their devices, which gives them the ability to stay in control over their networks.

8.2 Outlook

The Community System model and the security architecture are open enough to be applied in many domains, as presented below.

8.2.1 Communities of patients

A Community System could be applied in the domain of personal health care. Communities could organize a so called ”virtual patient record”, where the main stakeholders,
such as relatives, doctors, insurance company and employer, could get different views on the health of a patient. For example this could be statements from the doctor on the health of the patient, the results after certain examination, etc, which are published in the patient’s community. We can look at the community not only on human-to-human level, but also on machine-to-machine level. For example the hospital information system, which is registered in the patient community collects data from medical devices that monitor the patient at home. The hospital system would then be able to notify the doctor in case of abnormal values of the measured property of the patient, etc.

8.2.2 Virtual office

A Community System could be applied between enterprises as well. Companies could build communities with their clients for better collaboration (Customer Relation Management systems). In such a space they can communicate on a project and share content, which could be either in the office, or at home. Even when people are traveling they can download the desired content from the community, if there is an Internet available. With the light Community Client (browser), the access to the community content enables people to work everywhere on any machine.
Bibliography


<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>Asset Browser (user role)</td>
</tr>
<tr>
<td>ACE</td>
<td>Access Control Entry</td>
</tr>
<tr>
<td>ACL</td>
<td>Access Control List</td>
</tr>
<tr>
<td>AH</td>
<td>Authentication Header</td>
</tr>
<tr>
<td>AP</td>
<td>Asset Provider</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APC</td>
<td>Asset Publishing Component</td>
</tr>
<tr>
<td>AS</td>
<td>Authentication Server</td>
</tr>
<tr>
<td>ASC</td>
<td>Asset Server Component</td>
</tr>
<tr>
<td>AW</td>
<td>Asset Writer (user role)</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>CC</td>
<td>Community Client</td>
</tr>
<tr>
<td>CGAS</td>
<td>Community Gateway Application Server</td>
</tr>
<tr>
<td>CP</td>
<td>Control Point</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relation Management</td>
</tr>
<tr>
<td>CS</td>
<td>Community System</td>
</tr>
<tr>
<td>CSA</td>
<td>Community System Application</td>
</tr>
<tr>
<td>DES</td>
<td>Data Encryption Standard</td>
</tr>
<tr>
<td>DIT</td>
<td>Directory Information Tree</td>
</tr>
<tr>
<td>DN</td>
<td>Distinguished Name</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System</td>
</tr>
<tr>
<td>DRM</td>
<td>Digital Rights Management</td>
</tr>
<tr>
<td>DSS</td>
<td>Digital Signature Standard</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>GRE</td>
<td>Generic Routing Encapsulation protocol</td>
</tr>
<tr>
<td>GwS</td>
<td>Gatewayed Services</td>
</tr>
<tr>
<td>HGW</td>
<td>Home Gateway</td>
</tr>
<tr>
<td>HSC</td>
<td>Home Security Console</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HyperText Transfer Protocol over Secure Socket Layer</td>
</tr>
<tr>
<td>IBM</td>
<td>International Business Machines</td>
</tr>
<tr>
<td>ID</td>
<td>IDentifier</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IKE</td>
<td>Internet Key Exchange</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IP(v4/v6)</td>
<td>Internet Protocol (version 4/ version 6)</td>
</tr>
<tr>
<td>IPSec</td>
<td>Internet Protocol Security (protocol suite)</td>
</tr>
<tr>
<td>J2EE</td>
<td>Java 2 Platform, Enterprise Edition</td>
</tr>
<tr>
<td>KDC</td>
<td>Key Distribution Center</td>
</tr>
<tr>
<td>LDAP</td>
<td>Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LDAPS</td>
<td>Security version of the Lightweight Directory Access Protocol</td>
</tr>
<tr>
<td>LDIF</td>
<td>LDAP Data Interchange Format</td>
</tr>
<tr>
<td>MAC</td>
<td>Message Authentication Code</td>
</tr>
<tr>
<td>MD4</td>
<td>Message Digest algorithm, number 4 (RFC 1320)</td>
</tr>
<tr>
<td>MIT</td>
<td>Massachusetts Institute of Technology</td>
</tr>
<tr>
<td>NTP</td>
<td>Network Time Protocol</td>
</tr>
<tr>
<td>OASIS</td>
<td>Organization for the Advancement of Structured Information Standards</td>
</tr>
<tr>
<td>oPAL</td>
<td>Offline Published Asset List</td>
</tr>
<tr>
<td>PAL</td>
<td>Published Asset List</td>
</tr>
<tr>
<td>PKI</td>
<td>Public Key Infrastructure</td>
</tr>
<tr>
<td>RC4</td>
<td>Rivest Cipher #4 (Ron Rivest is the &quot;R&quot; in RSA)</td>
</tr>
<tr>
<td>RFC</td>
<td>Request for Comments (IETF document)</td>
</tr>
<tr>
<td>RSA</td>
<td>algorithm by Rivest, Shamir, and Adelman</td>
</tr>
<tr>
<td>SAML</td>
<td>Security Assertion Markup Language</td>
</tr>
<tr>
<td>SC</td>
<td>Security Console</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SP</td>
<td>Service provider</td>
</tr>
<tr>
<td>SSH</td>
<td>Security Shell</td>
</tr>
<tr>
<td>SSL</td>
<td>Security Socket Layer</td>
</tr>
<tr>
<td>SSTC</td>
<td>Security Service Technical Committie</td>
</tr>
<tr>
<td>TCP</td>
<td>Transport Control Protocol</td>
</tr>
<tr>
<td>TGS</td>
<td>Ticket Granting Server</td>
</tr>
<tr>
<td>TGT</td>
<td>Ticket Granting Ticket</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol</td>
</tr>
<tr>
<td>UDDI</td>
<td>Universal Description Discovery and Integration</td>
</tr>
<tr>
<td>UPnP</td>
<td>Universal Plug and Play</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VITELS</td>
<td>Virtual Internet and Telecommunications Laboratory of Switzerland</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WS</td>
<td>Web Services</td>
</tr>
<tr>
<td>WSDL</td>
<td>Web Services Description Language</td>
</tr>
<tr>
<td>X.500</td>
<td>directory service (ITU standard)</td>
</tr>
<tr>
<td>X.509</td>
<td>digital certificate (ITU standard)</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>XMPP</td>
<td>eXtensible Messaging and Presence Protocol</td>
</tr>
</tbody>
</table>
X.509 certificates

A widely used standard for defining digital certificates from the International Telecommunication Union (ITU). The initial version of X.509 was published in 1988, version 2 was published in 1993, and version 3 was proposed in 1994 and published in 1995. Issuer and subject unique identifiers were introduced in Version 2, Extensions in Version 3. Version 3 addresses some of the security concerns and limited flexibility that were issues in versions 1 and 2. The current X.509 standard is an ITU recommendation (the ITU jargon for standard) approved in March 2000 and there are three "corrigendums" (changes), the last one from April 2004. X.509-in-the-large (not the data-format) also includes standards for Certificate Revocation List (CRL) implementations.

The structure of a X.509 v3 certificate is as follows:

- Certificate
- Version
- Serial Number
- Algorithm ID
- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Subject Public Key Info
  - Public Key Algorithm
  - Subject Public Key
- Issuer Unique Identifier (Optional)
- Subject Unique Identifier (Optional)
- Extensions (Optional)
• Certificate Signature Algorithm

• Certificate Signature

Authentication in X.509 can be carried out using either secret-key techniques or public-key techniques; the latter is based on public-key certificates. The standard does not specify a particular cryptographic algorithm, although an informative annex of the standard describes the RSA algorithm. ITU recommendations are non-binding and as a result, companies have implemented the standard in different ways. For example, both Netscape and Microsoft use X.509 certificates to implement SSL (see section 3.2) in their Web servers and browsers but an X.509 Certificate generated by Netscape may not be readable by Microsoft products, and vice versa.

The IETF defined a profile for use with X.509 and much more (PKIX). An elaborate overview can be obtained from http://www.ietf.org/html.charters/pkix-charter.html. A detailed description of the various fields in the X.509 certificate and an interesting list covering known problems in existing implementations can be found in Peter Gutman’s ”X.509 Style Guide” at http://www.cs.auckland.ac.nz/~pgut001/pubs/x509guide.txt The home pages of Peter Gutman (at http://www.cs.auckland.ac.nz/~pgut001/) are also worth an exploration (that is, if you have plenty of time). Some more profile formats are described in http://www.cypherpunks.to/~peter/part2.pdf (pages 44-47), but there are many, many more.
The aim of the Needham-Schroeder protocol [26] is to establish a mutual authentication between two parties that want to communicate. This protocol has been designed for authenticated communication in computer networks. A precondition for running the protocol is the existence of an authority, called Authentication Server (AS), in which the communicating parties are registered. The registration consists of the AS getting an identity of the participants in the network and exchanging a shared secret with them.

Figure C.1 presents the establishment of authenticated communication between Alice and Bob, who are registered at the AS. The protocol takes five steps to reach the goal.

Let us assume that Alice wants to set up a secure connection to Bob, thus she initiates the protocol. First Alice sends a request to the AS containing her identity, the identity of Bob and a challenge $C_A^1$. The AS responds with a message containing the secret key $K_{A,B}$, which later Alice shares with Bob, and a ticket $K_{B,AS}(K_{A,B})$. The challenge, known as nonce, is a random number chosen from a big set. It is used to uniquely relate two messages to each other, thus preventing replay attacks.

Once Alice receives the ticket from the AS, she starts the secure channel with Bob by sending message 3, which contains challenge $C_A^2$, encrypted with the key $K_{A,B}$ and the ticket $K_{B,AS}(K_{A,B})$. Bob decrypts the ticket, extracts the shared secret key $K_{a,b}$ and returns the response $C_A^2 - 1$, together with $C_B$ to Alice, all encrypted with $K_{A,B}$. Finally, at step 5, Alice responds to Bob with a $C_B - 1$.

![Figure C.1: The Needham-Schroeder authentication protocol](image)
D.1 Registration of devices

Motivation

The de-centralized model for assets sharing suggests that assets are stored at devices belonging to a user or a group of users. The information about the assets and references to them are stored on the Community Server (CS). In that respect the CS needs to establish a trusted relation with the asset storage device. This section elicits the steps to register a device from a user as well as from a technical perspective.

Prerequisites

- The user has a device D, which has the potential to publish assets in the Community System. The user contacts the CS for the first time with this device.
- The device D is Internet enabled.
- The device D has a globally unique public/private key pair. This pair is stored in a secret cache.
- The device D has a globally unique identifier. The identifier is derived from the hash of the public key.
- The device D has a manufacturer-installed, or potential to accommodate, version of the Asset Provider (AP) software.

The device may also be re-registered (reset) in the Community System.

User perspective

Main scenario

1. The user buys new device, which has a manufacturer-installed version of AP application.
2. The user enables Internet access to the device.
3. The user starts the Asset Publisher Component (APC) of the AP.
4. The user logs in the CS.
5. The user starts registration procedure of the device at the Community System.

6. The device D confirms registration.

An alternative scenario is when the user installs the AP application manually.

1. The user buys new device, which has **no** installation of the AP software.

2. The user installs version of the AP software on the device.

   Continues as the previous scenario.

**Technical scenario**

The user starts the AP registration procedure, which includes the following steps (see also Figure D.1):

1. The AP generates certificate request.

2. Establish secure communication with the CS (HTTPS). The CS contact details are obtained from the configuration file. In case the device has the CS certificate it verifies that it communicates with the correct CS.

3. The AP sends certificate request to the CS, its ID and a signature of the ID. The request contains also the user credentials.

4. The CS authenticates the user.

5. The CS receives the device certificate request.

6. The CS verifies the signature.

7. The CS signs the certificate request.

8. The CS registers the device in the system by entering the following information:

   (a) Device identity.

   (b) Device certificate.

   (c) Device IP address/URL

9. The CS sends back to the device the signed certificate.

10. The AP receives its new certificate.

11. The AP stores its certificate.

**Remarks**

The idea in the scenario is that the device contacts the CS in order to be registered and establish trust relation. The registered device is invisible for the services provided by the CS until the device requests and/or offers a service (e.g. assets sharing). The AP application is responsible to publish assets and manage the necessary
D.1. REGISTRATION OF DEVICES

Figure D.1: Device Registration Scenario

information at the client side.

**Post conditions**

- The user has a device, with installed version of the AP software, which registered at the CS.
- The device is ready to publish and serve assets.

**Threat analysis**

This scenario is very important for the trusted environment of the CS. The possible threats are:

- The CS is not what it claims to be - the AP software should have the CS server certificate, or the user accepts the certificate he/she receives from the CS.
- Stolen device identity - if malicious software tries to add or delete information at the CS it should have the sufficient user credentials.
- Eavesdropping - encrypted channels prevent eavesdropping.
D.2 Asset publication

Motivation

Important feature of the Community System is to allow users to publish assets. This scenario elicits the steps that need to be implemented. Asset Publication scenario includes two parts: Select assets and Publish assets in a community. The first scenario is performed online and/or offline. The second action is performed only when the device is online. These two actions are described in different sub-scenarios.

D.2.1 Select Assets

Prerequisites

- The user is registered in at least one community.
- The user has a device, which is registered at the CS. This implies that the device has the AP software installed.
- The AP software is able to publish assets.
- The AP software is able to cache the user’s community view locally.
- The user has Asset Writer role in his/her community.

User scenario

1. The user starts the Asset Publishing Component (APC) of the AP.
2. The user logs in the system.
3. The user chooses "Asset Publication" tab. There the user is able to enter the community path and select an asset for publication.
4. The user selects an asset.
5. The user publishes the asset by pressing a "Publish" button.

Technical Scenario

The user has selected the asset and its place in the community. On pressing the "Publish" button the APC executes the following sequence actions:

1. The APC obtains the community path and the local path of the asset.
2. The APC generates a unique identifier of the asset. The identifier is a hash of the user ID, community path, the device ID and the local path concatenated as a string.
3. The previous information plus the generated identifier are stored in the structure of the local Community view. When the device is online the Community view is synchronized with the Community System.
Remarks

The collected information is stored in a temporary pool called Offline Published Asset List (oPAL) that is part of the Community view. Whenever the device is online the information from oPAL is sent to the CS, after which the actual Published Asset List (PAL) is updated. The latter step is described in the Publish assets in community scenario.

Post conditions

- The user has selected a number of assets to be published in his/her communities asset trees.
- The APC has stored in the local community view the information about the assets that are going to be published.

Threat Analysis

The main threat in this scenario is non-authorized access to the local community view. Several solutions are possible:

- Allow only online assets publication - users should be online when they publish assets in the CS. It eliminates the efforts for the synchronization of the local community view with the one at the CS (the copy at CS rules).
- Sign the community view file with user secret. It implies that AP software should store user password, which is not desired behavior.

D.2.2 Publish assets in community

Prerequisites

- The user is registered in at least one community.
- The user has a device, which is registered at the CS. This implies that the device has an AP software installed.
- The AP software provides functionality to publish assets with its Asset Publishing Component (APC).
- The AP software stores locally a copy of the user’s Community view. Part if this view is an Offline Published Asset List (oPAL) that stores offline published assets.
- The user has selected assets to be published.
- The device is connected to Internet.
- The user has Asset Writer role.

User perspective
1. The user logs in the CS via the APC.
2. The APC notifies the user about the offline selected assets.
3. The user checks the list before he/she actually publish them in the community.
4. The user presses the "Publish" button.

**Technical Scenario**

1. The *Select asset* scenario has been invoked.
2. The APC connects to Community System.
3. The application sends a list with the asset to the CS. The list includes asset name, user id, device id, community path, and asset identifier.
4. The CS obtains the list, authenticates the user and inserts the assets in the community repository.
5. The CS acknowledges the publications to the APC.
6. The APC updates the Published Asset List.
7. The published assets are deleted from the Offline Published Asset List.

**Remarks**

This scenario depicts how published assets are populated from the temporary buffer (oPAL) to CS and the "official" PAL. This step is performed only when AP software is online. In case the user has published (selected) assets offline, then the AP software offers the user functionality to confirm online publication.

**Post conditions**

- The user has published references to his/her assets in his/her communities.
- The assets’ content is stored on user’s device.
- The AP software has information about the published assets stored at its PAL.

**Threat Analysis**

- The CS is not what it claims to be - the AP application has the CS server certificate and is able to detect malicious Community Systems.
- Storage device is not what it claims to be - the CS has public key of the device and is able to detect malicious devices.
- Encrypted channels protect from eavesdropping.
D.3 Access to assets

Motivation

An important feature of the system is to allow community members to access the shared assets within the community. The access should be secure, authorized and transparent. This scenario presents the steps necessary to implement this behaviour.

Prerequisites

- A community called "Photographer" exists.
- Steve and Jan are registered in the "Photographer".
- Steve has a Community Client (CC) installed.
- Jan visited the "Keukenhof".
- Jan has a device that is registered at the Community System (CS). The device has AP application installed on it.
- Jan published pictures from the trip.

User perspective

1. Steve starts the Community Client.
2. Steve establishes secure connection with the CS.
3. Steve logs in the CS.
4. Steve opens the "Photographer" community.
5. Steve browses assets tree of the community.
6. There Steve finds pictures from the trip to "Keukenhof".
7. Steve clicks on the link of a picture. The picture is loaded and visualized in an appropriate application program (e.g. picture viewer).

Technical scenario

On click over the link to a picture the following actions take place:

1. The CC is directed to the CS\(^1\).
2. The CS receives the asset request.

\(^1\)The URL behind the picture looks like: https://community_system_address:port/asset_application/asset_handler?asset_community_path=value
3. The CS creates an assertion in the form of ticket. The ticket contains the CS name, the asset identifier, and the request time. The CS takes the following steps to accomplish the ticket:

   (a) Obtains information about the asset - asset identifier, storage device reference, asset name.
   (b) Creates the ticket structure.
   (c) Encrypts the ticket with its private key to confirm the source.
   (d) Extracts the public key of the device (via the device reference).
   (e) Encrypts the ticket again - this time with the public key of the device.
   (f) Encodes the ticket for HTTP transportation (Base64 and URLEncode).

4. The CS "glues" the ticket to the URL of the device\(^2\).

5. The CS returns the generated URL to the CC.

6. The CC is redirected to the device URL.

7. The CC and the Asset Provider (AP) establish secure connection (HTTPS, based on the AP certificate).

8. The AP device handles the request in the following steps:

   (a) Decrypts the ticket - decode from the transportation wrappers, decrypted with the AP private key, decrypt with the CS public key. The usable output from these operations verifies the source of the ticket.
   (b) Ticket validation - checks the ticket time stamp for freshness.
   (c) Check existence of the assets - the AP checks its Published Asset List (PAL) for the existence of the asset, and its local path.

9. The AP send the content of the asset.

Post conditions

- Steve has transparently accessed an asset.

Threat Analysis

- Secure channels prevent eavesdropping - attacks are prevented in case all participants are aware with whom they communicate.

- Access to CS repository is protected via username/password and different levels of access to data (role based access). The intruder should steal the username/password pair of a user in order to penetrate the system. Apart from the conventional ways (physical treat, steal, etc), the intruder might sniff packages on the network (further it refers to channels security).

\(^2\)The URL looks like: http://device_address:port/asset_handler_application/asset_handler?ticket=encrypted_ticket&etc_parameters
D.4 Device presence

Motivation

Veri often an asset storage device is offline for certain period of time. Thus we should consider a mechanism that informs the users about the availability of the assets published in the system.

Prerequisites

- The user in this scenario is registered in at least one community.
- The user has a device D, which is registered at the Community System (CS). This implies that the device has the AP application installed.

User perspective

1. The user connects his/her asset provider device to Internet.
2. The device status in the Community System is changed to online.

Technical Scenario

1. The device sends signal to CS to authenticate itself. The message contains the device identifier and signature of this identifier, after signing it with its private key.
2. The CS verifies the signature.
3. The CS updates connection information of the device.
4. The CS notifies the device if it needs to update its server certificate.

In case the device need to update the certificate the device executes the sign certificate procedure from the registration scenario.

Post conditions

- The asset storage device is online.
- The CS has update information the contact information of the asset storage device.

Threat Analysis

- The main threats in this scenario are when either the CS or the AP is impersonated. In both cases the benevolent party recognizes malicious party using the public key of the other.
- Encryption of the connection prevents eavesdropping.
E.1 Secure Device Actions

UPnP Security has a DeviceSecurity service which is added to UPnP devices when the device actions need security. The service requires a number of actions which are divided in two categories: actions invoked by Control Point (CP) and actions invoked by Security Console (SC). The first category is related to the protection of the services offered by the device, while the other is related to the administration of the security in the device.

The actions in the first category are invoked by CPs and SCs, and are the following:

1. GetPublicKeys - Returns the public keys corresponding to the private keys held by the device.

2. GetAlgorithmsAndProtocols - Retrieves a description of the algorithms and protocols supported by the device.

3. GetACLSizes - Returns the current and total allocation sizes for ACLs and related storage.

4. CacheCertificate - Invoked by CPs to deliver their Authorization and Named Group certificates to the device (optional action).

5. SetTimeHint - Optional action, supported by devices that have no local time source, but need time to process ACLs.

6. GetLifetimeSequenceBase - Retrieves the current value of the LifetimeSequenceBase state variable. The variable is a non-repeating value used to prevent replay for TakeOwnership and SetSessionKeys actions.

7. SetSessionKeys - Allows a CP to establish symmetric keys for subsequent signed or encrypted operations.

8. ExpireSessionKeys - Permits a CP to inform a device explicitly that it no longer intends to use a session.

9. DecryptAndExecute - Provides secrecy of a SOAP message by tunneling it inside an action argument.

The actions in the second category are invoked only by SCs.
APPENDIX E. UPNP SECURITY

1. *TakeOwnership* - Permits a Security Console to claim ownership of an unowned, security-aware device.

2. *GetDefinedPermissions* - Returns a list of permissions defined by the manufacturer for the device.

3. *GetDefinedProfiles* - Returns the set of manufacture defined profiles and their definitions.

4. *ReadACL* - Returns a device’s ACL and the current version of that ACL.

5. *WriteACL* - Replaces the existing ACL of the device with a new ACL, assuming that the caller knows the correct version of the ACL being replaced.

6. *AddACLEntry* - The provided ACL entry is added to the device’s ACL, assuming that it is not already present, is correctly formatted and has memory in which to be held. The action must be invoked by authorized entities (owners and possibly delegates).

7. *DeleteACLEntry* - Deletes and ACL entry. The action must be invoked by authorized entities (owners and possibly delegates).

8. *ReplaceACLEntry* - If successful it replaces one ACL entry with a new value. The action must be invoked by authorized entities (owners and possibly delegates).

9. *FactorySecurityReset* - Returns the device to initial conditions, once the device is disconnected from the network (or, perhaps, on next power-up). The action must be invoked by an owner of the device.

10. *GrantOwnership* - Adds a co-owner to devices, which support multiple owners. The action must be invoked by an owner of the device.

11. *RevokeOwnership* - Removes a co-owner from the device owner list. The action must be invoked by an owner of the device.

12. *ListOwners* - Returns a list of the key hashes of all owners of the device.

Additional information about those actions can be found in [7].
E.2 Security Console Actions

A Security Console (SC) implements a number of actions used to communicate with Control Points and other Security Consoles. The actions are as follows:

1. **PresentKey** - Accepts an offered key has from a control point on the network in order to do "discovery" of control points without forcing them to become UPnP Devices and announce their existence by SSDP.

2. **GetNamedList** - Returns an XML element, encoded as a string, containing all of the names defined by the SC.

3. **GetMyCertificates** - Retrieves and returns the full set of certificates being held for a Control Point. The action is invoked by Control Points.

4. **RenewCertificate** - The certificates used in UPnP are issued for a short period of time. This action is invoked by a Control Point to renew its certificates.

Additional information about those actions can be found in [8].

E.3 UPnP certificates

UPnP Security supports two types of certificates - Authorization and Named Definition. The first one is equivalent to a signed ACL entry. The difference with ACL is that the issuer has to be specified as well as the target device. In addition the certificate very often has a validity period. E.g. an entry looks like this:

```
<entry>
  <subject>
    <hash><algorithm>SHA1</algorithm>
    <value>dRDPBgZzTFq7Jl2Q2N/YNghefj8=</value></hash>
  </subject>
  <access xmlns:mfgr="."><mfgr:display/> <mfgr:play/></access>
</entry>
```

while an authorization certificate looks like:

```
<cert us:Id="." xmlns="urn:schemas-upnp-org:service:DeviceSecurity:1">
  <issuer>
    <hash><algorithm>SHA1</algorithm>
    <value>Gd48BqQzAMPn4FkWnFslMMdxSG4=</value></hash>
  </issuer>
  <subject>
    <hash><algorithm>SHA1</algorithm>
    <value>dRDPBgZzTFq7Jl2Q2N/YNghefj8=</value></hash>
  </subject>
  <tag>
    <device>
      <hash><algorithm>SHA1</algorithm>
      <value>2jmj75rSw0yVb/vlWAYK/YBwk=</value></hash>
    </device>
  </tag>
</cert>
```
APPENDIX E. UPNP SECURITY

The Name Definition certificates add one (Control Point) key (or one whole group) to a named group. A key is added by referring to its hash. A named group is added by referring to its name.

subject describes the Control Point, while define tag describes the group. Depending on the implementation the define tag may contain more than one group.
The Asset Provider and the Community System use a number of configuration and data structure files. Those files are described in this appendix together with an example.

F.1 Published Asset List

The Published Asset List (PAL) is used by the Asset Provider to store the published assets. It provides the mapping between the published files and their local paths. It consists of asset entries which have publisher, asset identifier and local path. An example of a PAL.xml is the following:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<assetlist>
  <asset>
    <userID>tomcat</userID>
    <assetidentifier>m1Goz4I+BtzVTVKI6iLrEkAw4=</assetidentifier>
    <localpath>C:\jakarta-tomcat-working\P6100399.jpg</localpath>
  </asset>
  <asset>
    <userID>Chavdar</userID>
    <assetidentifier>UuuBKUPfqapR7gEWdQbYJQGDY=</assetidentifier>
    <localpath>C:\jakarta-tomcat-working\P6100399_old.jpg</localpath>
  </asset>
  <asset>
    <userID>chavdar5</userID>
    <assetidentifier>1dz+P6EgvHbmsfhkWSkqG00I=</assetidentifier>
    <localpath>C:\jakarta-tomcat-working\ffun_img.jpg</localpath>
  </asset>
</assetlist>
```

A PAL.xml has the following DTD file:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT assetlist (asset*)>
<!ELEMENT asset (assetreference, localpath, userID)>
<!ELEMENT assetreference (#PCDATA)>
<!ELEMENT localpath (#PCDATA)>
<!ELEMENT userID (#PCDATA)>
```
F.2 Offline Published Asset List

Offline Published Asset List (oPAL) is a data structure part of the community view at the Asset Provider. The oPAL stores the changes of the community view regarding assets. It consists of ready to publish asset entries, which have publisher, action, path in community, asset identifier, localpath, device identifier, asset name and friendly description. An example of a oPAL.xml is presented below:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<assetlist>
    <asset>
        <userID>chavdar5</userID>
        <action>add</action>
        <communitypath>cn=ffun_img.jpg, cn=assets,ou=Philips</communitypath>
        <assetidentifier>1dz+P6EgvHbmPSfuJkWSkq00l0I=</assetidentifier>
        <localpath>C:\jakarta-tomcat-working\ffun_img.jpg</localpath>
        <deviceID>kMu118s5kuawmR5YV7O+UAEpKMA=</deviceID>
        <assetName>ffun_img.jpg</assetName>
        <description/>
    </asset>
</assetlist>
```

The DTD file of a oPAL.xml is:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT assetlist (asset*)>
<!ELEMENT asset (userID, action, communitypath, assetidentifier, localpath, deviceID, assetName, description)>
<!ELEMENT userID (#PCDATA)>
<!ELEMENT action (#PCDATA)>
<!ELEMENT communitypath (#PCDATA)>
<!ELEMENT assetidentifier (#PCDATA)>
<!ELEMENT localpath (#PCDATA)>
<!ELEMENT deviceID (#PCDATA)>
<!ELEMENT assetName (#PCDATA)>
<!ELEMENT description (#PCDATA)>
```

F.3 Community System Configuration file

The Community System uses the following Configuration.properties file:

The name of the CS

```text
communityName=PhilipsGWS
```

Properties of the security cache of the CS.

```text
keyStoreFileName=C:/jakarta-tomcat/conf/communityServerKeyStore.kstr
keyStorePassword=community
communityKeyAlias=tomcat
```
communityKeyPassword=community
communityCertAlias=tomcat

Miscellaneous parameters: the URL of the asset handler at the Community Gateway Application Server (CGAS); the temporary folder that stores received certificate requests and generated new certificates; the algorithm used to encrypt and decrypt the ticket; the type of the repository.

assetHandlerURL=https://pc67240478.ddns.htc.nl.philips.com:8443/gws/assetRequestHandler
attachments_dir=../webapps/gws/WEB-INF/attachments
ALGORITHM=RSA/NONE/NoPadding
RepositoryType=LDAP

F.4 Asset Provider Configuration file

Both applications of the Asset Provider share the same Configuration.properties file, which is presented below:

The security identifier of the device (the hash of the device public key).

deviceID=kMu118s5kuawmR5YV7O+UAEpKMA

Paths to files which the AP components use. oPAL.xml is in the root directory of Asset Publisher Component (APC).

application_root=c:/community
pal_filename=C:/jakarta-tomcat/webapps/assetserver/WEB-INF/PAL.xml
tomcat_root=C:/jakarta-tomcat
opal_filename=./oPAL.xml

Properties of the security cache of the AP.

keyStoreFileName=c:/community/assetProvider.keystore
keyStorePassword=community
communityCertAlias=communitycert
assetProviderAlias=tomcat
assetProviderKeyPassword=community

Miscellaneous parameters: the URL of the Community Gateway Application Server (CGAS); the CS name; the algorithm used to encrypt and decrypt the ticket; the validity time of the ticket.

communityURL=https://pc67240478.ddns.htc.nl.philips.com:8443/community/servlet/AxisServlet
communityName=PhilipsGWS
ALGORITHM=rsa/none/nopadding
toleranceTimeout=30000
**Certificate Authority (CA)** - an authority in a network that issues and manages security credentials and public keys for message encryption and signature verification.

**Digital certificate** - consists of the public key and the identity of an entity, rendered unforgeable by digitally signing the entire information with the private key of the issuing Certificate Authority (CA).

**Key Distribution Centre** - KDC is a central point in a security architecture that shares secrets with all participants in the secure realm. KDC issues session secure keys when two parties in the realm want to establish secure connection.

**Private Key** - value - known only to one party - that can be used to decrypt encrypted messages, issue digital signatures and compute the corresponding public key.

**Public Key** - a value that can be used effectively to encrypt messages and verify digital signatures.

**Public Key Infrastructure (PKI)** - a system of digital certificates and Certificate Authorities that verify and authenticate the validity of each involved party.

**Ticket** - encrypted message containing sensitive information, such as session secret keys.
Curriculum Vitae

Chavdar Bachvarov was born in Pleven, Bulgaria on the 3th of August 1979.

In the period between 1986-1998 he obtained his primary and high school education.

In 1998 he studied 1 year in Technical University of Sofia, Bulgaria, at the Faculty of Computer Systems and Control. Even though his progress was very good, he decided to change the program with 4-year bachelor programm with the Faculty of Mathematics and Informatics at Sofia University "St.Kliment Ohridski". The major was in Computer Science and Mathematics. The bachelors degree was obtained in 2003.

Since 2003 the author started a 2-year Master of Science programme in Software Engineering at the Faculty of Electrical Engineering, Mathematics and Computer Science in Delft University of Technology, the Netherlands. In September 2004 he started his internship with Philips Research, Eindhoven. The result of the internship evolved in this MSc thesis. The work was closely supervised by Wim van der Linden and Rik Willems (SwA, Philips Research) as well as Leon Moonen (SE, Tu Delft).