Final Thesis

Flexible Certificate Management for Secure HTTPS Client/Server Communication

by

Jing Zhang

LITH-IDA-EX--05/071--SE

2005-09-22
Final Thesis

Flexible Certificate Management for Secure HTTPS Client/Server Communication

by

Jing Zhang

LITH-IDA-EX--05/071--SE

2005-09-22

Supervisor: Andreas Weber
Hewlett-Packard GmbH

Examiner: Nahid Shahmehri
Abstract

Certificate management is a crucial element in PKI implementations, which includes certificate generation, distribution, storage, and revocation. Most of the existing research has been focusing on the security aspect or the functionality and the structure of certificate management systems. Very little has looked at the actual user requirements for the system and how users can use the system conveniently and practically, which is actually a very important factor for the whole security system to work properly and to be widely accepted.

In this thesis we have designed a framework that provides a flexible certificate management for different security levels according to user requirements and situations, and with a user-friendly interface. A certificate management system CSA (Certificate Server Adapter) is implemented for HP OpenView Operations for Windows (OVO/Windows), which is a management software product provided by Hewlett-Packard. The CSA helps OVO/Windows to provide secure HTTPS client/server communication. Tests show that it offers a good enough security for all situations without compromise and, at the same time, the best convenience and flexibility are achieved. However, the CSA can be further improved to have a lifetime management of the created certificates, an enhanced user interface, and an API to plug-in other PKI solutions.
Acknowledgement

First of all, I would like to give my deepest thanks to my supervisor Andreas Weber, for all the support, help, and valuable suggestions he has given to me throughout the work. And I want to thank the Hewlett-Packard Boeblingen OVO/Windows R&D team for giving me this opportunity to do my thesis in the group and offering me unlimited help and a very pleasant working environment. And thank you, to the many other friends at HP, not only for technical support, but also for the kindness and friendship.

And of course I would like to thank Nahid Shahmehri for being my examiner, who made my thesis work outside the university possible. You have given me so many insightful and valuable comments. Thanks very much!

Thanks to all of you, who have spent the last half year with me. It really means a lot to me. I’ll cherish all the memories and experience. Without you, this thesis had not been possible.
# Contents

1 **Introduction** ................................................................................................ 1  
1.1 Overall Problems ...................................................................................... 1  
1.2 Motivations ............................................................................................. 1  
1.3 Objectives ............................................................................................... 2  
1.4 Outline ................................................................................................... 2  

2 **Background** ............................................................................................ 3  
2.1 Theoretical Background ............................................................................ 3  
  2.1.1 Computer Security and Cryptography................................................... 3  
  2.1.2 HTTPS .............................................................................................. 4  
  2.1.3 Public Key Infrastructure (PKI) ............................................................ 5  
  2.1.4 Certificate Management .................................................................... 8  
2.2 Related Works ......................................................................................... 8  

3 **Analysis** ................................................................................................ 11  
3.1 Goals .................................................................................................... 11  
3.2 The Application System ........................................................................... 11  
  3.2.1 Introduction ..................................................................................... 11  
  3.2.2 Architecture ..................................................................................... 11  
  3.2.3 Installation Scenarios ........................................................................ 12  
3.3 Goals Revisited ....................................................................................... 15  
3.4 Discussion .............................................................................................. 16  
  3.4.1 Certificate Deployment ...................................................................... 16  
  3.4.2 Node Mapping .................................................................................. 21  
  3.4.3 Operating in a Multiple-Server Installation Environment .................. 21  
3.5 Selected Approach .................................................................................. 24  

4 **Design** .................................................................................................... 27  
4.1 Architecture ........................................................................................... 27  
  4.1.1 Microsoft Technologies ...................................................................... 28  
4.2 Basic Assumptions and Definitions ............................................................ 30  
  4.2.1 Certificate Request Types .................................................................. 30  
  4.2.2 Auto-Granting Certificate Requests ..................................................... 30  
  4.2.3 Node Mapping .................................................................................. 31  
  4.2.4 Certificate States .............................................................................. 32  
  4.2.5 Multiple-Server Environment .............................................................. 34  
4.3 Flow Chart ............................................................................................. 34  
4.4 Use Cases .............................................................................................. 36  

5 **Implementation** ..................................................................................... 43  
5.1 Implementation ...................................................................................... 43  
  5.1.1 Developing Environment .................................................................... 43  
  5.1.2 Database and Model .......................................................................... 43  
  5.1.3 COM Interfaces and Classes ............................................................... 44  
5.2 User Interface Design ............................................................................. 45  
5.3 Command Line Tools ............................................................................. 45  
  5.3.1 The Tool ovowcsa ............................................................................. 46  
  5.3.2 The Tool ovowcsacm .................................................................... 48
List of Figures

Figure 2-1 SSL in the network architecture ................................................................. 4
Figure 2-2 X.509 Certificate Contents ........................................................................ 7
Figure 3-1 OVO/Windows Architecture .................................................................... 12
Figure 3-2 Single-server installation ......................................................................... 12
Figure 3-3 Multiple-server installation: Geographic Distribution ......................... 13
Figure 3-4 Multiple-server installation: Expert Center ........................................... 14
Figure 3-5 Multiple-server installation: Win/UNIX Specialist .................................. 14
Figure 3-6 Multiple-server installation: Backup Server .......................................... 15
Figure 3-7 OVO/Windows PKI Environment ............................................................. 16
Figure 3-8 Manual certificate deployment ................................................................. 18
Figure 3-9 Manual certificate deployment with one-time installation key .............. 19
Figure 3-10 Remote manual certificate deployment ............................................... 20
Figure 3-11 Remote automatic certificate deployment .......................................... 21
Figure 3-12 Multiple-Server Environment Certificate Management - Expansion .... 22
Figure 3-13 Multiple-Server Environment Certificate Management – Merge Step 1:
  Two separate environments ............................................................................... 23
Figure 3-14 Multiple-Server Environment Certificate Management – Merge Step 2: CA
  certificate of Server 2 is added to the trusted certificates of Server 1 and CA
  certificate of Server 1 is added to the trusted certificates of Server 2 ................. 23
Figure 3-15 Multiple-Server Environment Certificate Management – Merge Step 3:
  Updated of trusted certificates for nodes in both environment 1 and 2 ............. 24
Figure 4-1 CSA Architecture Overview ................................................................... 28
Figure 4-2 COM and COM interface ....................................................................... 28
Figure 4-3 Node-mapping Flow Chart .................................................................... 32
Figure 4-4 Certificate State Diagram ..................................................................... 33
Figure 4-5 Flow Chart for Processing a New Certificate Request ......................... 35
Figure 4-6 Flow Chart for Processing an Incoming Notification ......................... 36
Figure 4-7 Use Case Diagram ................................................................................ 37
Figure 5-1 Database table for node-mapping ......................................................... 43
Figure 5-2 CSA Class Diagram and COM Interfaces ............................................. 44
Figure 5-3 The CSA User Interface ....................................................................... 45
Figure 5-4 Listing pending requests with the command line tool ovowcsa ............ 47
Figure 5-5 Issuing a new certificate with the command line tool ovowcsacm ...... 48
Figure 5-6 Test Cases ......................................................................................... 50
Chapter 1

Introduction

This thesis was written as the final part of my studies towards a master degree in Computer Science at Linköpings universitet, Sweden. All the work was performed in the Hewlett-Packard GmbH OpenView Boeblingen Lab, Germany.

1.1 Overall Problems

Security has become one of the big research challenges and is receiving increasing attention throughout the entire IT industry [1]. With the rapid growth of the Internet, and the new ways of doing business and managing business it enables, maintaining confidentiality, privacy, authentication, and so on, is becoming an urgent issue. One approach to resolve this issue is through the use of cryptographic technology, for example, SSL (Secure Socket Layer) [2], which is a popular cryptographic protocol for authenticating the server (and the client) and creating a secure communication channel. PKIs (Public Key Infrastructures) [3] have been developed and deployed to address many security-concerned applications, such as secure distributed systems, SSL-based electronic commerce and execution of downloaded files. Secure communications over the Internet are based on certificates and certificates are the building blocks of the PKI system. Certificate management is therefore very important.

Certificate management is the administration of certificates. This activity includes generation, distribution, storage and verification of certificates [4]. To get the whole infrastructure work, certificate generation and distribution is above all the most essential part, and needs to be well taken care of. There are several different ways of managing certificates. The traditional approach is to deploy the certificate personally or manually. This method is protected but has a number of disadvantages. The prime disadvantages are geographical restrictions. This technique is impractical in the age of the Internet, and very inconvenient. There is also remote transparent certificate deployment, which uses TCP, a connection-oriented end-to-end protocol. This approach is of course much faster and more efficient. But does it provide enough security for the user? And how can the management of certificates be as easy to use, and as flexible to cater for different situations?

1.2 Motivation

HP OpenView Operations for Windows (OVO/Windows) [5] is a management software solution provided by Hewlett-Packard. It is used for monitoring big computing environments with hundreds or thousands of servers running different operating systems (especially all the Windows and UNIX flavors) and business applications (e.g. databases, ERP\(^1\), and mail servers). There is one central management server that monitors the whole environment with small agents on every managed node that are configured to survey the hardware, operating system, and applications according to

\(^1\) Enterprise Resource Planning, management information systems.
the setup of the managed node. The agents collect performance data and send alarms to the management server in case anything goes wrong or any problems are predicted.

At the moment, the communication between the management server and the managed nodes is done via remote procedure calls (DCE [6]). The team is working on a solution that will allow all communications over HTTPS [7] (SOAP [8]). The Agents and the management server will identify each other by the exchange of certificates for nearly every interaction. Thus, a Certificate Management component is needed for the system PKI, to provide security (Authentication, Data integrity, and Confidentiality), and a user-friendly interface.

1.3 Objectives

The objectives of this thesis are to design a framework of certificate management and to implement a certificate management system for the OVO/Windows management software system. The goal is to provide a flexible management for different security requirements and situations, and with a user-friendly interface.

1.4 Outline

The rest of the thesis is organized as follows:

Chapter 2 provides a background to the thesis. Theoretical foundations together with related works on certificate management are presented.

Chapter 3 gives an analysis of the problem to be solved, including a discussion of the application system (OVO/Windows), requirements, and approaches.

Chapter 4 and 5 describe the design, implementation, and testing of the certificate management system, respectively, followed by the results and evaluation.

Finally Chapter 6 is a summary of the thesis work where conclusions and the future work are presented.
Chapter 2

Background

This chapter provides a background for this thesis. It starts with the theoretical foundations including discussion of computer security and cryptography, HTTPS (HTTP over Secure Socket Layer), Public Key Infrastructure (PKI), and certificate management. Then related work on certificate management in both the academic field and the commercial world are described and examined.

2.1 Theoretical Background

2.1.1 Computer Security and Cryptography

The problem of computer security has become more and more prominent, especially with the growing interest in networks and distributed systems in the age of Internet explosion [9]. Network security ensures that only authenticated end users can access the network resources [10].

The three major principles of network security are: Confidentiality, Integrity, and Availability.

- Confidentiality: the assurance of data privacy, that is, only authorized persons can get protected information. Confidentiality often requires physical protection for unencrypted data, encryption of physically unprotected data, access control in the computer systems, and correct programming of the protection.

- Integrity: the assurance of non-alteration, that is, the data has not been undetectably changed. Data integrity often requires access control, relevant protocols to detect unauthorized changes, digital signatures or other cryptographic checksums to enable the detection, and physical protection of points where protection software can be removed.

- Availability: the assurance of the data to be available for those authorized to access it. Availability requires undamaged computer and storage media, tolerable environment for the computer operation, and functioning communications and programs.

In addition, we want Accountability for anyone breaking this, thus we need Authentication of user identity. Auditing of logs (and keeping the right logs) is crucial for accountability, finding the vulnerability that enabled an attack, finding what to restore and how to restore it after an attack, and a general knowledge of what is going on.

Cryptography secures information by enciphering it in an unreadable format with various cryptographic algorithms, such as RSA and International Data Encryption Algorithm (IDEA) [10]. It uses cryptographic algorithms to encipher and decipher
information. Cryptography is a security tool which converts a network problem into a key management problem.

2.1.2 HTTPS

HTTPS means HTTP over Secure Socket Layer (SSL). It was developed by Netscape to encrypt and decrypt transmitted data over the Internet [10]. HTTPS is an Internet protocol that provides a SSL layer of security. HTTPS uses SSL and HTTP to protect the communication channel between the client and the server on a network. When HTTP is used to access the data on the Internet, HTTPS provides strong authentication [10].

SSL refers to an intermediate (security) layer between the transport layer and the application layer. It is layered on top of a connection-oriented and reliable transport service, such as provided by TCP/IP (Transmission Control Protocol/Internet Protocol), and to provide security services for TCP-based application protocols, such as HTTP (HyperText Transfer Protocol), IMAP (Internet Message Access Protocol), LDAP (Lightweight Directory Access Protocol). Figure 2-1 shows the architecture, with the SSL protocol layered between the application layer and TCP/IP layer.

![Figure 2-1 SSL in the network architecture](image)

When SSL is used, the client and the server start by performing a handshake [11]:

- The client and the server agree on what cryptographic and hashing algorithms to use.
- The client receives a certificate from the server and validates it.
- Both agree on a symmetric encryption key.
- The encrypted communication starts.

The handshake may also include a client certificate to let the server authenticate the client, but that step is optional. After the handshake is done, control is passed to the original handler, who now talks plain HTTP over the encrypted channel [11].

The SSL protocol enables SSL server authentication, SSL client authentication, and an encrypted SSL connection between a client and a server. In SSL server authentication, a client confirms the identity of the server using the SSL-enabled client software, which uses public-key cryptography techniques. The SSL-enabled client software validates the certificate and the public ID of a server and verifies that they are issued by a certificate authority (CA) that is on a list of the client’s trusted CAs. In SSL client authentication, a server confirms the identity of a client, using the SSL-enabled server software, which uses public-key cryptography techniques. The SSL-enabled server software also validates the certificate and the public ID of a client and verifies that they have been issued by a CA that is on a list of the server’s trusted CAs. An encrypted SSL connection between a client and a server provides confidentiality to the transmitted data. The sender encrypts the data and the receiver decrypts the data, using the SSL-enabled software [10].
The SSL protocol is by far the most pervasive security protocol for the Internet in general [11], and there have been more and more implementations of HTTPS.

2.1.3 Public Key Infrastructure (PKI)

PKI (Public Key Infrastructure) consists of standards for handling public keys. There are several definitions [12]:

The comprehensive system required to provide public-key encryption and digital signature services is known as the public key infrastructure (PKI). The purpose of a public key infrastructure is to manage keys and certificates.

-Entrust Inc.

A public-key infrastructure (PKI) consists of the programs, data formats, communications protocols, institutional policies, and procedures required for enterprise use of public-key cryptography.

-Office of Information Technology, University of Minnesota

In its most simple form, a PKI is a system for publishing the public-key values used in public-key cryptography. There are two basic operations common to all PKIs:

Certification is the process of binding a public-key value to an individual organization or other entity, or even to some other piece of information such as a permission or credential.

Validation is the process of verifying that a certificate is still valid.

How these two operations are implemented is the basic defining characteristic of all PKIs.

-Marc Branchaud

To summarize, a Public Key Infrastructure (PKI) is a security infrastructure based on certificates and public key cryptography as its security components, and a coherent implementation of technical, procedural, and legal components aimed at allowing relying parties (those who rely on certificates when participating in a transaction) to trust the binding between a public key and the name of the owner. It has a set of security services that enable the use and management of public key cryptography and certificates [13]. Three important notions in PKI are: Public Key Cryptography, Digital Certificates, and Certificate Authority. We describe each below.

Public Key Cryptography

Public key cryptography is a commonly used cryptographic method, which uses an asymmetric key pair (a public key and a private key). The key used for encryption is different from the one used for decryption. Public key cryptography requires the key owners to protect their private keys while their public keys are not secret at all and can be made available to the public. The computation algorithm relating the public key and the private key is designed in such a way that an encrypted message can only be decrypted with the corresponding other key of that key pair, and an encrypted message cannot be decrypted with the encryption key (the key that was used for encryption) [13]. Public key cryptography is one of the essential components of modern network security for many reasons. First, it scales better than symmetric cryptography, as each participant needs only one pair of keys to communicate with everyone else (as opposed to sharing a key with each participant). Second, it partly solves the key distribution problem associated with symmetric cryptography. Finally, public key cryptography, by associating a unique and private key with each participant,
provides a way for secure protocols to link actions to individuals, thereby enabling digital signatures and non-repudiation [14].

**Digital Certificates**

It is important to ensure the security of a public key to avoid security breaches related to impersonation and key modification. Therefore, a data integrity mechanism is required to ensure that a public key that is modified does not go undetected. However, data integrity mechanisms alone are not sufficient to guarantee that the public key belongs to the claimed owner. A mechanism is required which binds the public key with some globally trusted party that can ensure the identity and authenticity of the public key [15]. The desired mechanism should accomplish the following two goals:

- Establish the integrity of the public key
- Bind the public key and its associated information to the owner in a trusted manner

In the PKI environment, digital certificates accomplish these goals. A digital certificate is a credential (in digital form) in which the public key of the individual is embedded along with other identifying data. That credential is encrypted (signed) by a trusted third party or certificate authority (CA) who has established the identity of the key owner. By “signing” the certificate, the CA establishes and takes liability for the authenticity of the public key contained in the certificate and the fact that it is bound to the named user [12]. Certificates ensure that only the public key for a certificate that has been authenticated by a certifying authority works with the private key possessed by an entity. This prevents someone from using a fraudulent public key to impersonate the public key owner, and the message cannot be repudiated.

**The X.509 Standard**

The SSL protocol does not depend on a particular format for the public key certificates it exchanges. As far as SSL is concerned, a certificate is just an arbitrary set of bytes [17]. Practical PKI deployments and implementations, however, depend heavily on the specifics of those certificates. One particular international standard is widely accepted as the appropriate format for public key certificates. That standard is from the International Telecommunications Union (ITU), and is universally known by its ITU specification number: X.509 [17].

Certificates that conform to the latest X.509 standard can contain as many as 11 different fields. Their order in the certificate corresponds to the illustration in Figure 2-2. Note though that the field names in the Figure are not the same as the names in the X.509 standard. A brief description of each field: Version describes the version of the encoded certificate; Serial Number is a positive integer assigned by the CA to each certificate, which is unique for each certificate issued by a give CA; Algorithm Identifier is the identifier for the algorithm which is used by the CA to sign the certificate; Issuer identifies the entity who has signed and issued the certificate; Period of Validity is the time interval during which the CA warrants that it will maintain information about the status of the certificate; Subject identifies the entity associated with the public key stored in the subject public key field; Subject’s Public Key is used to carry the public key and identify the algorithm with which the key is used; Issuer Unique ID and Subject Unique ID are present in the certificate to handle the possibility of reuse of subject and/or issuer names over time; Extensions is a SEQUENCE of one or more certificate extensions; Signature contains the value of the digital signature computed upon the above fields.
After a digital certificate is obtained, the entity can use it to communicate with recipients of information in the following manner: The subscriber digitally signs the message with his or her private key to ensure message integrity and its own authenticity and sends the message to the recipient. The recipient, after receiving the message, verifies the digital signature with the subscriber’s public key and queries the global directory database, which is a software application storing information about a network and its resources, to check the validity of the subscriber’s digital certificate. The Global directory database returns the status of the subscriber’s digital certificate to the recipient. The transaction is completed only if the certificate is valid.

**The Certificate Authority (CA)**

The Certificate Authority (CA) is a trusted third party that authenticates entities and issues digital certificates. A CA can be public or private, it can be internal or an organization, national, or international. For this process to be secure, the CA’s public key must be trustworthy and well-known. A CA must also perform the necessary due diligence to verify that individuals or entities are in fact who they say they are, before a digital certificate is issued to an individual or entity [18].

To ensure that PKI is implemented successfully, a CA has two principal roles to perform [15]:

- Ensure that the public key that is associated with a private key belongs to the person specified in the public key. If the public key belongs to a person other than the one specified, the entire purpose of PKI is defeated. Therefore, the CA performs the important task of issuing certificates and revoking them if they are compromised.

- Establish its credibility, so that all parties can trust the CA. A CA uses its private key to sign certificates. The public key corresponding to this private key is stored in a certificate. Further, this certificate enables a user to establish a trust relationship with a CA.

The CA is the most important component, as it is the CA that signs certificates and revocation information. In order to limit risk, it is standard to deploy a root CA, which is the trust anchor of the whole infrastructure. The root CA signs certificates of other “signing” CAs, which may then be used to sign subscriber certificates or to sign certificates for CAs lower in the hierarchy of trust. The root CA limits the depth of the resulting hierarchy by defining a parameter known as the basic constraint [14].

To be able to communicate with each other in a PKI, users need to obtain certificates from the CA by sending certificate requests. According to certain policies, CA will decide whether to issue the certificate and grant or deny the certificate request. Whenever two or more PKI clients want to communicate securely, they need to
validate each other based on digital certificates, and negotiate the various encryption, authentication, and data integration algorithms [15]. Thus, applications can achieve confidentiality, integrity, authentication, and non-repudiation by using a PKI. As the PKI becomes interoperable, scalable, and generally accepted, it has compelling justifications for products and enterprises to incorporate it for wider use. But it is not that easy, especially during implementation. There are problems like operational and technical incompatibilities, underestimated scope, and immature technology [12].

2.1.4 Certificate Management

To make the use of a security service with certificates possible in a computer networks, certificate management turns out to be the most critical element [19]. A successful operation of a PKI relies largely on effective key and certificate management. Certificate management may be difficult because it includes each individual participating in the PKI. In a large organization with potentially thousands of users, this can be demanding.

Certificate management generally includes creation, distribution, storage, and verification of certificates [19]. Major activities involved are certificate request, certificate generation, certificate installation, certificate revocation, and certificate validation. The certificate management process starts with the user requesting a certificate. The system administrator will then decide whether or not grant the request and issue the certificate. Once it is issued and generated, a certificate remains valid for a certain period of time. This is known as the certificate lifetime, which is determined by the CA policy. Certificates can become invalid before their expiration date. For example, if a private key is compromised, or even if the compromise is suspected, it can cause a certificate to become invalid before expiry.

2.2 Related Work

PKIs and certificate management are not new topics, but they are still attracting a lot of attention in both the academic field and the commercial world.

Liu, Cant, Ozols, and Henderson [19, 4] proposed a mathematical model and an approach to the formalization of certificate management systems. A top-down hierarchical structure was discussed to perform all functions needed for efficient certificate administration, such as certificate issuing, certificate revocation, changes of keys, and certificate verification.

Kapidzic and Davidson [20] discussed the structure of certificate management system, and presented a certificate management system that implements all necessary functions for the administration of certificates. They specified new roles and responsibilities for certification authorities, and extended the functions of the certificate management system for the storage and retrieval of certificates, to make it functionally complete and immediately operable.

Oppliger, Greulich, and Trachsel [21] proposed an architecture for a distributed certificate management system that can also be used for group-based access controls. The system can be used to build and operate a scalable PKI, and distribute the operational task among several people.

Ungureanu [22] proposed a mechanism for the specification of a wide range of certificate management policies and for their enforcement. The author argued that the traditional way of certificate management, which is performed manually or by trusted agents, is too expensive, error prone, and might impose a high degree of strain on a system. The certificate management could be viewed as any other on-line service. But certificate management in this paper is not specific for PKIs.
As we have discovered, most of the research are focusing on the structure or architecture for certificate management systems, the efforts are spent on the functionalities the system has, and how secure it can be, and how the system can be implemented. Very few have considered the actual user requirements of the system; how the system can be used conveniently and practically. These are the critical factors for the system to be used and accepted widely. Only Ungureanu [22] has rendered the concern for the efficient use of the system and made the system sort of automatic by using a mechanism to enforce certification and revocation policies, which still did not solve the different situations the users might be in. Sometimes it may not be secure enough to have the automatic operations. So it can not be considered as a complete solution and a flexible system.

Currently there are also an increasingly large number of commercial certification service providers which are offering their services to the general public, most of them based on X.509. Among the major players are VeriSign, Baltimore Technologies, Entrust Technologies, and Thawte Consulting (acquired in February 2000 by VeriSign) [23]. In recent years a host of companies joined them in their race to capture the certificate market (either products or services). Also, major corporations including American Express, AT&T, Canada Post, CompuSource, Equifax, Hewlett-Packard, IBM, Lotus Development, Microsoft, Motorola, Netscape, and Novell all support the X.509 digital certificate standard [23].

A quick survey shows that most of the services and products are used to secure files, e-mail, WLAN (wireless LAN), and e-commerce. This thesis work will be focusing on secure client/server communication in software management applications.

With the background of the thesis and theoretical foundation being introduced, in the next chapter, we are going to analyze the problem to be solved.
Chapter 3

Analysis

This chapter presents the analysis of the problem to be solved, including discussions of application systems, requirements, and approaches.

3.1 Goals

As a crucial part of PKIs, certificate management has received considerable attention [24]. Most of the existing research has been focusing on the security of the system, and the functionality and structure of the certificate management system. However, very little has looked at the actual user requirements for the system, how users can use the system conveniently and practically, which is actually a very important factor for the whole security system to work properly and to be widely accepted.

The goal of this thesis is to design of a framework for certificate management and the implementation of a certificate management system for a management software product. The goal is to provide flexible management for different security requirements and situations, and with a user-friendly interface.

3.2 The Application System

The management software product that this thesis is working with is called HP OpenView Operations for Windows (OVO/Windows). In this section, a brief introduction of the application system is given.

3.2.1 Introduction

HP OpenView is one of the flag software products of HP. It is an integrated, holistic approach that dynamically links business and IT to enable a new level of insight and control. Currently about 70% of the Internet devices are managed by OpenView applications. It is used by 100% of the Fortune 50 companies, most of the large US-based ISPs, and with totally 135,000 installations [25].

OVO/Windows is a product in the OpenView family. It is a management software solution to monitor big computing environments with hundreds or thousands of servers running different operating systems (Windows, UNIX, and Linux) and business applications (e.g. databases, ERP, and mail servers). It provides powerful functionalities such as Windows-based event management, proactive performance monitoring, automated alerting, reporting, and graphing.

3.2.2 Architecture

OVO/Windows utilizes a three-tier architecture made up of a management server, management consoles, and managed nodes, as shown in Figure 3-1. The
management server in the middle of the figure is a central repository for configuration, management, and event information. Operators and administrators interact with the management server through the management console on top. A web console is provided for operators to use for message management functions. The managed nodes at the bottom can be computers or network components. Intelligent OpenView Operations agents are installed in the nodes and are configured to survey the hardware, operating system, and applications according to the setup of the managed nodes. The agents collect performance data and send alarms to the management server in case anything goes wrong or any problems are predicted.

Figure 3-1 The OVO/Windows Architecture

3.2.3 Installation Scenarios

To better understand the application system, different installation scenarios are presented below. Basically they are classified into two categories: single-server installation and multiple-server installation.

3.2.3.1 Single-Server Installation

The single-server installation is the typical installation of an OVO/Windows software system. Figure 3-2 shows the structure of this type of installation. There is one central management server in the middle that monitors the whole environment with small agents on every managed node at the bottom.

Figure 3-2 Single-server installation

3.2.3.2 Multiple-Server Installation

Besides the typical single-server installation, there might be a demand for a multiple-server installation to manage the environment more effectively and efficiently, especially for large-scale environments. For a multiple-server installation, there are several servers, and the whole computing environment is configured to have one
management server to communicate with another management server (server-based flexible management), or to have one agent to communicate with several servers according to different criteria (agent-based flexible management). There are several scenarios where such a configuration is very helpful, which are discussed next.

**Geographic Distribution**

A worldwide company can have a management server on each continent. To balance the load between management servers and to provide follow-the-sun support, messages can be synchronized between management servers, while agents only report to their local management server (server-to-server message forwarding). Or agents could also be configured to send their messages to different management servers based on time. Figure 3-3 shows the communication configuration between management servers and agents for this case. The management servers are configured to synchronize messages among each other. The dotted line means the agent is configured to send the messages to another server.

![Figure 3-3 Multiple-server installation: Geographic Distribution](image)

**Expert Center**

Agents can be configured to send database related messages to the database service center management server. Other application messages are sent to the application center management server. All other messages are sent to the local management server. This scenario could also be built up with server-to-server message forwarding: the agents send all their messages to the local management server and the management server forwards these messages to the database service center or the application center based on message attributes. Figure 3-4 shows the communication configuration between management servers and agents for this case. The agents are configured to send different types of messages to the respective service center. Also the management server can forward different types of messages to the respective service center.
Companies that use OVO/UNIX as the main management servers may also want to add an OVO/Windows management server to their environment in order to capitalize on the strengths of OVO/Windows in managing their Windows systems (e.g. Exchange Smart Plug-In, Active Directory Smart Plug-In, and Windows Management Instrumentation Policies). Or companies that are mainly Windows shops, but have also a considerable amount of UNIX systems that they would like to manage with OVO/UNIX. In this scenario each management server can manage and configure its own type of systems and messages are exchanged and synchronized between the OVO/Windows and OVO/UNIX management servers (server-to-server message forwarding). Figure 3-5 shows the communication configuration for this case. The two management servers are configured to synchronize messages. And the dotted lines means the nodes are also configured to send messages to different servers.

For high availability, server-to-server message forwarding can also be used to always have two management servers in synchronization, so that the backup server can take the role of the primary server in case of a failure. Figure 3-6 shows the
communication configuration for this case. The primary management server and the secondary management servers are configured to be synchronized.

![Diagram of multiple-server installation: Backup Server]

Figure 3-6 Multiple-server installation: Backup Server

The above installation scenarios show how OVO/Windows application systems are implemented and how the communication between clients and servers, and between servers in different situations is accomplished.

3.3 Goals Revisited

At the moment, the communication between the management servers and the managed nodes in OVO/Windows is done via remote procedure calls (DCE). A new solution is being implemented that will allow all requests to be performed over HTTPS (SOAP). The agents and the management servers will identify each other by the exchange of certificates for nearly every interaction.

There are basically three situations where requesting and granting certificates are needed to initiate communications:

1. The management server adds a new managed node. The managed node needs a certificate from the server in order to communicate.
2. The first communication between two management servers.
3. The first communication between a managed node and another management server, instead of the local management server.

A PKI (Public Key Infrastructure) environment has been previously implemented with fundamental security components related to certificates. Figure 3-7 shows the PKI environment and how it is setup for OVO/Windows systems. (On the figure, all certificates are in place.) On the top of the figure is a management server system hosts the Certificate Server, which contains the needed certificate authority (CA) functionality. On the lower part of the figure are the managed nodes (Node1, Node2, NodeX) connected to the management server in the network. On every managed node, there is a Certificate Client, holding a node certificate that has been signed by the Certificate Server with the CA private key. The management server system has a node certificate as well in order to proof its identity, just like every other node does. In addition, every system has a list of trusted root certificates (with at least one root certificate as shown in the figure). The trusted root certificates are used to verify the identities of the communication parties; a party will only be trusted if it has a root certificate that has been signed by a trusted CA.
The security component in the application system consists of two parts: the Certificate Server on the management server and the Certificate Client on the managed nodes. The Certificate Server is responsible for operating a certification authority and executing certificate related operations like issuing new certificates, exporting certificates, and distributing certificates. The Certificate Client is responsible for tasks like creating and sending certificate requests to a Certificate Server, and receiving and processing a corresponding response. So the actual execution is done by the security component, and it provides an API for getting certificate requests as well as granting and denying specific requests.

What is still missing here is a certificate management system as the interface between the security component and OVO/Windows system, for managing and administrating the distributed public-key infrastructure. As stated earlier, this certificate management system should provide a good secure feature, and also a good usability.

3.4 Discussion

3.4.1 Certificate Deployment

Certificate deployment can be done in several ways. The traditional way is to grant the certificate personally or manually. This is protected and can be considered quite
safe, but it has several disadvantages. One of the major drawbacks is obviously geographic restriction. It is time-consuming, costly, and very inconvenient. For the OVO/Windows application system, which is normally used in a large computing environment, and very likely is geographically distributed, this approach is definitely not practical and is not efficient.

We can also use the automatic remote approach, which will grant incoming certificate requests automatically. This way is of course very fast and very convenient, so that it appears almost transparent to users since users do not need to interact with the handling of certificates. But if we do not have any control on certificate requests and issuing that means we compromise the security concerns, which surely is not what we want out of the PKI implementation and the need for certificates.

Having considered these two extreme approaches, we realize that actually there is no ideal way to guarantee both at the same time, that is, very secure and very fast. Instead of providing a compromise, which is neither too secure nor too fast, we propose a flexible way according to the user’s request of security level. If the user has a strong security need, she can select the very secure way of deploying certificates. Otherwise she can select the faster and more convenient way. Thus, we can offer good enough security for all situations without compromise. The following four ways will be offered to deploy a certificate:

1. **Manual certificate deployment**

   This is the traditional manual way, which might be appropriate for highly secure environments. In this method, a public/private key pair is generated on the Certificate Server and then transported to the node system. No direct communication between the server and the client is required. Figure 3-8 is a visual representation of this approach.

   - The node certificate and the corresponding private key for a specific node are generated on the server, and a password can be used to encrypt the created data, as in the figure step 1.

   - Then the created files that contain the certificate, the corresponding private key and the trusted certificates are transported to the node system via a portable medium or any other existing secure channel, as in the figure step 2. Here we assume the courier is fully reliable, which means it should only be transported by trusted persons, and access to the medium has to be protected to prevent a malicious attacker.

   - After the installation, as in the figure step 3, the data on the portable medium and the previously created file on the server system should be deleted or wiped out for security reasons.
2. Manual certificate deployment with one-time installation key

This approach also requires some manual steps. This way has the advantage that the private key never leaves the system it belongs to. However, it requires a working communication link between the server and the client. Also the certificate request and the signed certificate need to travel over the network before the node gets its certificate. Figure 3-9 is a visual representation of this approach.

- The server generates a one-time installation key, together with some additional information in a file, and a password can be used to encrypt the created key, as in the figure step 1.

- The file is then transported to the node system, as in the figure step 2.

- On the node, a certificate request will be initiated and encrypted with the installation key, as in the figure step 3.

- The server will then automatically grant the certificate request that is encrypted with the right installation key. The signed certificate will be sent back to the node, as in the figure step 4.
3. Remote manual certificate deployment

In the remote deployment approach, the concept of a configurable secret key is introduced. It is provided by the functionality of the security component, and is used to encrypt the network traffic between the node system and the server system until the node system has a valid certificate. Until then, this shared secret key is also used for the authentication of the two communication parties. If the message was encrypted with the correct key, the other side trusts the sender.

In the remote manual certificate deployment method, the public/private key pair together with a certificate request are generated on the node system. The certificate request will be encrypted with a configurable secret key and sent to the certificate server system. The server decrypts the certificate request and then the user evaluates the request to decide whether to grant or deny it. This eliminates the need to transport files manually; everything will be done remotely. But the granting of certificates still needs manual interaction from the user and the certificate request encrypted with the secret key is sent over the communication link before the node gets its certificate. This does not provide full security, and is therefore not recommended for environments that ought to be highly secured. Figure 3-10 is a visual representation of this approach.

- On the node, a public/private key pair is generated; also a certificate request is initiated, as in the figure step 1.

- The certificate request is encrypted with a secret key and sent to the server system, as in the figure step 2.

- The user will evaluate the certificate request, and decide whether grant it or deny it, as in the figure step 3.

- If the request is granted, the certificate request is signed by the Certificate Server, and encrypted with the secret key and sent back to the node system, as in the figure step 4.
If the request is denied, the server system will send a message to the node that the request has been denied.

4. Remote automatic certificate deployment

Sometimes we may have good reasons for granting certificates automatically, without any consideration or manual operations. For example, in the OVO/Windows application system, if the server just adds a new node for management and the node is known to the server, then the server is expecting a certificate request from the node for the communication between the two systems. Or if the certificate request comes from a node that is known, for example, within an IP address range, or belongs to certain node groups, then the user can be sure that it can be granted, and automatic granting will be a good option. Here of course we provide the most convenient way for deploying certificates, but then consequently it leaves some holes in the security. The use of this option and the auto-grant configuration will be decided by the user. Figure 3-11 is a visual representation of this approach.

The steps of this approach are similar to the remote manual certificate deployment (see Figure 3-10), but instead of the user manually evaluating the certificate request, the system will check whether the request fulfills the auto-grant criteria and grant or deny the request.
As discussed above, we want to provide four different ways of deploying certificates, which are at different levels of security, and at different degrees of convenience with regard to use and at different speeds.

### 3.4.2 Node Mapping

When we deploy a certificate, especially for the remote certificate deployment, either the user (manually) or the system (automatically) needs to decide whether to grant or deny the certificate request. The decision should be based on where the certificate request comes from, and whether the node which sends the certificate request can be trusted.

One of the goals for this thesis is to have a user-friendly interface and to provide an easy-to-use system. A very imperative need is to have a straightforward view of the nodes which send the certificate requests. From the certificate request itself, the unique identifier for the node is some ID, which may not make sense for the user. So we need to get more information about the certificate request and the node, such as the IP address, host name, computer platform, and the operating system of the node.

Furthermore, to be used and integrated in OVO/Windows application system, the node which sends the certificate request has to be aligned with the nodes in the system database. Thus, a node-mapping from the certificate request to the node in the OVO/Windows node list is important, and the mapping must be based on the information we can get from the certificate request, that is, either the IP address or the hostname.

### 3.4.3 Operating in a Multiple-Server Installation Environment

There are two possible scenarios for managing certificates in a multiple-server installation environment, assuming that a Certificate Server (CS) will be installed on every management server.
a) The first scenario is called "expansion", which means there is one existing operating Certificate Server in the environment, and an additional Certificate Server needs to be installed, but the second server would not create a self-signed CA certificate but use the existing server’s CA certificate to sign its own CA certificate. In this case, the nodes and servers in this environment would still trust each other. See Figure 3-12 for a visual representation. Server 1 is the existing management server with an operating Certificate Server CA 1, and Server 2 is the second server. Node 1 and Node 2 are managed nodes of Server 1. Node 3 and Node 4 are managed nodes of Server 2. The CA certificate of Server 2 is signed by the CA certificate of Server 1.

![Figure 3-12 Multiple-Server Environment Certificate Management - Expansion](image)

b) The second scenario is called "merge", which means two existing managed environments are both having an operating Certificate Server and we then need to merge them into a single environment. This can be solved by adding the CA certificate from one server to the trusted certificates of the other, and the same thing the other way around. Then the managed nodes are triggered to get the updated list of trusted certificates from their certificate server. Now the nodes and servers in the merged environment can trust each other. Figure 3-13, Figure 3-14, and Figure 3-15 show the merging procedure step by step.
Figure 3-13 Multiple-Server Environment Certificate Management – Merge
Step 1: Two separate environments

Figure 3-14 Multiple-Server Environment Certificate Management – Merge
Step 2: CA certificate of Server 2 is added to the trusted certificates of Server 1
and CA certificate of Server 1 is added to the trusted certificates of Server 2
3.5 Selected Approach

Following the above analysis, we propose a certificate management system called the Certificate Server Adapter (CSA) for the OVO/Windows application system. The format of certificates and the method for their verification are defined in the X.509 standard. Therefore the focus of this thesis will be the administrative aspect of certificate management, and specifically the functionality of certificate request and certificate grant. Certificate requests are granted or denied automatically by the certificate server adapter, based on certain criteria, or manually by the user from the GUI.

The CSA should perform the following functions:

- Listen for an incoming certificate request
- Evaluate the certificate request
- Map the certificate request with the node from the node list in OVO/Windows
- Automatically grant the certificate request when possible
- Offer an interface for mapping the node sending the certificate request into the node list when needed
- Offer an interface for granting/denying the certificate request
- Offer an interface for configuring the rules for automatic granting of certificates
- Update the OVO/Windows database: change the certificate status for each node
- Listen for an incoming notification about a successful/unsuccessful certificate installation.

The design and implementation of the CSA is based on the following assumptions:

- The OVO/Windows server is up and running
- The Certificate Server and the Certificate Client are up and running
- The certificate request contains enough information so that it is possible to make a decision on the handling of the certificate request
- The Certificate Server API for getting certificate requests as well as granting and denying specific requests is available.
This chapter has a detailed analysis and discussion of the problem to be solved, and an approach is proposed. In the next two chapters, the design and implementation of the system CSA are presented.
Chapter 4

Design

This chapter describes the design of the CSA (Certificate Server Adapter) system, including the architecture, assumptions and definitions, detailed process flow, and use cases.

4.1 Architecture

The overall architecture of the system is shown in Figure 4-1. In the context of the OVO/Windows three-tier architecture, the CSA is a sub-component of the management server, hosted by the Security Server component, with the GUI as part of the Management Console. The Certificate Server and the Certificate Client are available, respectively, from the management server and the managed node.

The CSA will get a list of certificate requests from the Certificate Server, and pass grant or deny commands to it. The communication between the CSA and the Certificate Server is done via HTTPS (see middle of the figure, to the left). The CSA also maps certificate requests to nodes in the OVO/Windows node list, and stores the state of certificate requests in the Database on the left of the figure, via WMI (Windows Management Instrumentation).

The CSA GUI, on top of the figure, accesses the DCOM (Distributed Component Object Model, see section 4.1.1) interface of CSA on the server to get the list of certificate requests and show them to the user. It provides the interface for users to grant and deny certificate requests, and also the interface to map certificate requests to nodes in the node list of OVO/Windows.
4.1.1 Microsoft Technologies

Since OVO/Windows is based on the Windows platform, it uses many Microsoft technologies as its building blocks. This section will give a brief introduction to those relevant for this thesis.

**COM**

The Component Object Model (COM) [26] is a binary standard that lets objects operate together in a networked environment regardless of the language that they were developed in or the computers that store them. COM-based technologies include ActiveX Controls [27], automation, and OLE (Object Linking and Embedding) [28]. COM allows an object to expose its functionality to other components and to host applications. It defines how the object exposes itself and how this exposure works across processes and across networks. COM also defines the object's life cycle.

A COM object is a piece of compiled code that provides some service to the rest of the system. In COM, applications interact with each other and with the system through collections of functions called interfaces. An interface is a definition of an expected behavior and expected responsibilities. All OLE services are nothing more than COM interfaces. A COM interface is a strongly typed contract between software components to provide a small but useful set of semantically related operations (methods). Figure 4-2 shows the interaction between a client application and a COM object. The COM object offers three methods for the client application to access.
**DCOM**

Microsoft Distributed COM, or DCOM [26], extends the Component Object Model (COM) to support communication among objects on different computers-on a LAN, a WAN, or even the Internet. DCOM is the protocol that allows you to access a COM object on a remote computer. Using DCOM, OVO/Windows can be distributed.

DCOM handles low-level details of network protocols and uses the extensible security framework provided by Windows.

**WBEM**

Web-Based Enterprise Management (WBEM) [29] is an initiative that was started a number of years ago by a group of companies (BMC Software and Cisco among them) to provide a common method of accessing management data through any type of utility. These companies organized into the Distributed Management Task Force (DMTF) [30].

WBEM provides a common layer of management from which applications like OVO/Windows can pull data. This data, provided by the program interfaces such as Simple Network Management Protocol (SNMP) [31] or the Windows registry, resides in a WEBM repository called the Common Information Model (CIM) [32].

**CIM**

WBEM uses the Common Information Model (CIM, see WBEM above) to represent and to store managed objects. CIM is a data model, a conceptual view of the environment, which unifies and extends existing instrumentation and management standards such as SNMP and Desktop Management Interface (DMI) [33], using object-oriented constructs and design.

The core of CIM is platform-independent and supports platform-specific extensions. As with any other management architecture, WBEM needs a way of storing and retrieving management data. CIM is that storage and retrieval mechanism.

As mentioned above, CIM is the database for WBEM. It uses object constructs that allow for dynamic changes of the data structure and the establishment of relationships between objects in that structure. It is this customizable aspect of CIM that OVO/Windows has leveraged through its service model and by allowing administrators to create service definitions to represent their unique environments.

**WMI**

Windows Management Instrumentation (WMI) [34] is Microsoft’s implementation of the WBEM, which is an industry initiative to develop a standard technology for accessing management information in an enterprise environment. It defines a non-proprietary set of environment-independent specifications. These specifications allow management information to be shared between management applications (like OVO/Windows) that might run in both similar and dissimilar operating system environments.

WMI prescribes enterprise management standards and related technologies that work with existing management standards such as DMI and SNMP. Like WBEM, WMI complements these existing standards by providing a uniform model of the managed environment through which management data from any source can be accessed in a common way.

WMI uses the CIM and Internet protocols for providing management data as well as Microsoft’s object technology (COM) and communication technology (DCOM).
OVO/Windows utilizes the WMI at various places on the management server. The WMI can also be used as a data source on Windows-based managed computers.

4.2 Basic Assumptions and Definitions

4.2.1 Certificate Request Types

As discussed in Chapter 3, we have four types of certificate deployment according to the security level requested by the user. Based on these four types, two types of certificate requests are defined:

- **MANUAL_WITH_INSTALLATION_KEY**: With this type of certificate request, the certificate should be installed using a manually distributed installation key. In this case, if auto-grant is enabled, the certificate is automatically granted; otherwise, the user needs to decide whether to grant/deny the request using the GUI.

- **REMOTE**: With this type of certificate request, the certificate will be installed remotely. If auto-grant is enabled and configured, CSA first checks if the certificate request fulfills the auto-grant criteria. If so, the certificate is automatically granted. Otherwise, the user needs to decide whether to grant/deny the request using the GUI.

For certificates that should be installed completely manually, no certificate request is needed, so we do not have a certificate request type in this case. Also the installation procedure is not managed by the CSA system, since it is all manual. But once the certificate is installed, a notification should be sent to CSA, and the state of the certificate should be stored in the database.

4.2.2 Auto-Granting Certificate Requests

Granting certificate requests is a security sensitive task which should be performed with special care. A certificate request should only be granted by an administrator when he is sure that the request is valid. On the other hand, there are some situations that the certificate requests should be granted for sure. So for time-saving and for the convenience of users, CSA provides an Auto-grant mechanism for processing certificate requests.

There are four situations that are defined as possible for auto-granting:

a) Certificate request type is **MANUAL_WITH_INSTALLATION_KEY**.

b) Within Timeframe: The certificate request can be mapped to a node that was installed within a pre-defined timeframe. E.g. the agent installation on the node was triggered no longer than 30 minutes ago.

c) Within IP Address Range: The certificate request comes from a node with an IP address in the pre-defined IP address range.

d) Belonging to a Certain Node Group: The certificate request comes from a node that is a member of a certain node group that is pre-defined.

This auto-granting mechanism can be configured by the administrator to be enabled or disabled. If the administrator chooses to enable auto-grant, he can configure in what situation(s) and on what criteria the certificate requests can be auto-granted.
4.2.3 Node Mapping

In the OVO/Windows database, there is a node list for each management server. Before executing any grant/deny action on the incoming certificate request, the CSA will map the node which issues the certificate request to the node in the node list. The node-mapping can help the user to be aware of which node sends the certificate request, so that the decision of grant/deny can be made easily.

The unique identifier for nodes is contained in OvCoreID, which is created in a node before it sends the certificate request. However, normally the OvCoreID is not yet in the OVO/Windows database unless the certificate gets installed. Because of this, we have to map nodes by other means. From certificate requests, the host name can be intercepted, so we will use this attributes to map nodes.

- **Automatic mapping**: this is performed first

  1a) The node list is checked whether there exists an OvCoreID that is the same as the one in the certificate request. If found, mapping succeeds.

  1b) If the OvCoreID for the certificate request is not found in the node list, then check whether there exists a node in the node list which has the same host name and does not have an OvCoreID yet. If found, mapping succeeds.

- **Manual mapping**: if automatic mapping fails, the user has to map nodes manually using the GUI

  2a) The user selects a node which has the same host name as the one found in the certificate request, and which does not yet have an OvCoreID.

  2b) The user selects a node which has a different host name than the one in the certificate request, and which does not yet have an OvCoreID. (A warning window might pop up.)

  2c) In addition to the above two situations, the rest would be nodes with different OvCoreIDs. In these cases, mapping would fail; otherwise if we change or delete the OvCoreID of the node, the information about the node in the database would be lost. If mapping fails, there are two options:

  I. Add a new node to the node list which has the identity OvCoreID of the certificate request, and then do the mapping.

  II. Delete the current certificate request, change the OvCoreID on the corresponding managed node and then reissue a new certificate request.

Figure 4-3 shows the flow chart for node mapping, which is a visual presentation of the above description.
Certificate request intercepted

Map node by OvCoreID (1a)

Mapped?

Yes

Map node by hostname (1b)

Mapped?

Yes

Administrator manually selects node to map (2a)

Administrator discards the certificate request (II)

Map node by hostname (1b)

Mapped?

No

Administrator adds a new node to node list (I)

Certificate grant/deny logic

No

Yes

End

Figure 4-3 Node-mapping Flow Chart

4.2.4 Certificate States

Each node in the node list has a certificate state. Here, six certificate states are defined: UNDEFINED, PENDING, GRANTED, FAILED, DENIED, and INSTALLED. Each state represents the stage in which the certificate installation process is.

For OVO/Windows, the certificate state of a node is important so that the administrator has an overview of the PKI for the complete managed environment. Also, any managing activity such as policy deployment or message sending can only be done after the certificate is installed on the node. For this purpose, two states (UNDEFINED and INSTALLED) are sufficient, while the CSA system needs an additional state, PENDING, to distinguish between the certificate requests that have not been mapped to a particular node yet and the certificate requests that have been mapped to a node but which have not yet been granted or denied. Only when the certificate state is PENDING can the administrator grant/deny a certificate request. Also, to manage the certificates that are installed manually, an additional certificate state GRANTED is introduced. When creating a certificate to manually install on a managed node from the server side, the node certificate state is changed to GRANTED. So the administrator will not issue another certificate for that node, even though the
certificate is not yet installed. The states of FAILED and DENIED are provided for the user so that she will have better information about the node certificate. To summarize:

- **UNDEFINED**: Before the node is successfully mapped to the node in the node list, its certificate state is UNDEFINED.

- **PENDING**: After the node is successfully mapped to a node in the node list, its certificate state is PENDING, until a grant/deny action is performed.

- **GRANTED**: When a grant action is successfully performed on the certificate request, the certificate state is GRANTED, until the CSA gets the notification from the Certificate Server that the certificate has been successfully installed. Alternatively, if a certificate is issued to be manually installed on the node, the certificate state is GRANTED before the certificate is successfully installed.

- **DENIED**: When a deny action is performed on the certificate request, the certificate state is DENIED.

- **FAILED**: If for some reason the grant action fails, or the certificate could not be installed on the node, the certificate state is FAILED.

- **INSTALLED**: The certificate state becomes INSTALLED once the certificate is successfully installed on the node and the CSA gets the notification from the Certificate Server.

The state diagram of the certificate is shown in Figure 4-4, which describes the different states of the certificate of a node. The initial state of a certificate can be UNDEFINED, if it is a new certificate request (see left of the figure), or GRANTED, if it is a manually installed certificate (see upper right of the figure). The final state of a certificate is INSTALLED, FAILED, or DENIED.

At the moment, if the grant/deny operation fails, the Certificate Server component cannot restart the processing of the request. Instead, the certificate request will be discarded. A certificate request can also be discarded during the states of UNDEFINED, PENDING, and GRANTED. Since the discarded certificate request will no longer be managed, the certificate state of the node will be changed to UNDEFINED again, waiting for new certificate requests.

![Figure 4-4 Certificate State Diagram](image)
4.2.5 Multiple-Server Environment

In section 3.4.3 we discussed the two possible scenarios for managing certificates in a multiple-server environment, namely expansion and merge. In both scenarios, a CSA system will be installed on each management server/Certificate Server. The CSA will manage the certificate requests from its managed nodes.

In the scenario of expansion, signing the new management server’s CA certificate will be done manually, and it will not be managed by the CSA on the existing server since the management server is not in the node list. In the scenario of merge, the adding of the CA certificate from one server to the trusted certificates of another server would also be done manually, not via the CSA system.

For nodes that are managed by several servers, the certificate is installed on the node from the first server that succeeds to do it (not necessary the primary manager). Once the node has its certificate installed, it will not send another certificate request. If a second management server tries to grant/deny a certificate request for a node which has already a certificate installed, the grant/deny operation will fail. It can happen that a node’s certificate state in the OVO/Windows database on one management server do not represent the correct certificate installation state, for example, when changing the management server during the certificate installation process, or if management server crashes during the same process. This can be solved by introducing a mechanism polling the node certificate states and synchronize them.

4.3 Flow Chart

A high level flow chart of the functionality of the CSA for the processing of a new certificate request is presented in Figure 4-5. It starts with an incoming certificate request. The CSA then registers the certificate request and deletes duplicates if necessary. Then the CSA tries to map the node in the node list automatically. If the node could not be mapped, there is the GUI for the administrator to manually map the node or to add a new node to the node list. If the auto-mapping succeeds, the CSA checks the certificate deployment type. If it is manual with installation key and if auto-grant is enabled, the certificate request is granted. If the type is remote, and if auto-grant is enabled, the CSA checks whether the certificate request fulfills auto-grant criteria, if yes then the request is granted, otherwise it offers the GUI for the administrator to decide whether grant or deny the request.
Figure 4-5 Flow Chart for Processing a New Certificate Request
Besides the processing of the incoming certificate requests, the CSA also listens for the notifications about a successful/unsuccesful certificate installation/grant/deny. The processing of all the other notifications is presented in Figure 4-6. It starts with an incoming notification. Then, the CSA will intercept the notification. If it is a new request, the new request processing procedure will follow. If the notification is for grant failed or certificate installed, the CSA tries to find the node in the node list, and changes the certificate state for the node. If the node could not be found, an exception will be thrown. For the other notifications, the CSA does not perform anything.

![Flow Chart for Processing an Incoming Notification](image)

4.4 Use Cases

In software engineering, a use case is a technique for capturing the potential requirements of a new system or software change. Each use case provides one or more scenarios that convey how the system should interact with the end user or another system to achieve a specific business goal [35]. In this thesis, we use this technique to have a better understanding of the requirements of the CSA system before the actual implementation.
Use case 1: View Request

**Description:** This use case allows the user to view the list of certificate requests, with detailed information about the node which sends the certificate request, such as host name, IP address, platform, and the information about the certificate, such as request Id, and received time.

**Actors:** Administrator.

**Assumptions:** There are certificate requests that need to be processed. Otherwise, there will be no requests in the list.

**Success end condition:** None.

**Failure end condition:** None.

**Main success scenario:**
1. The actor accesses the GUI to view certificate requests.
2. The CSA triggers Certificate Server to get the list of certificate requests, together with the detailed information for each request.
3. The GUI of the CSA presents the information to the actor.

**Extensions:** None.

**Variations:** None.
Use case 2: Map Request to Node

**Description:** This use case allows the user to map a certificate request to a node in the OVO/Windows node list.

**Actors:** Administrator, CSA for automatic mapping.

**Assumptions:** The certificate request is in the state of UNDEFINED, FAILED, or DENIED.

**Success end condition:** The certificate request is related to a managed node in the OVO/Windows node list, and the certificate state is PENDING.

**Failure end condition:** The certificate request is not related to the existing managed nodes, or it is related to a node that has a different OvCoreID.

**Main success scenario:**
1. The administrator triggers the map node operation for a request via the GUI.
2. The administrator selects a managed node to which the request shall be mapped from a list of managed nodes.
3. The CSA ensures that the OvCoreID of the node is either empty or equal to the OvCoreID in the certificate request.
4. The CSA changes the certificate state of the node to PENDING, and updates it on the GUI.

**Extensions:** None.

**Variations:** If there is a node in the node list which has the same OvCoreID as the certificate request, or if the node has the same host name but does not have an OvCoreID, the CSA will automatically map the request to that node.

Use case 3: Un-map Node

**Description:** This use case allows the user to delete the existing node mapping.

**Actors:** Administrator.

**Assumptions:** The request has been mapped to a node but not processed yet, that is, the state is PENDING.

**Success end condition:** The certificate request is not related to the managed node in the OVO/Windows node list, and the OvCoreID of the node which was mapped before is changed to empty. The certificate request state returns to UNDEFINED.

**Failure end condition:** The certificate request is still related to the managed node, or the information stored for the node which was mapped before is not changed, or the state of the certificate request is not changed to UNDEFINED.

**Main success scenario:**
1. The actor triggers the node un-map operation for a request via the GUI.
2. The CSA removes the OvCoreID of the node which was mapped before.
3. The CSA changes the certificate state of the node to UNDEFINED, and updates it on the GUI.

**Extensions:** None.

**Variations:** None.

Use case 4: Add Node from Request

**Description:** This use case allows the user to add a new node from a certificate request in the node list of OVO/Windows.

**Actors:** Administrator.

**Assumptions:** The mapping node operation fails for the certificate request.

**Success end condition:** A new managed node is added in the node list; its fields are filled with node specific properties from the certificate request.

**Failure end condition:** The new managed node cannot be added because there is already a managed node with the same OvCoreID in the node list.

**Main success scenario:**
1. The actor triggers the add node operation for a request via the GUI.
2. The node specific properties are filled in with the information from the certificate request.
3. The actor adds additional information and saves the new node.
4. New node is stored in the database of the node list in OVO/Windows.

**Extensions**: None.

**Variations**: OVO/Windows has its own functionality for adding a managed node.

**Use case 5: Grant Request**

**Description**: This use case allows the user to grant a pending certificate request for a specific node.

**Actors**: Administrator, CSA for automatic granting.

**Assumptions**: The certificate request has been mapped to a node in the node list of OVO/Windows, and the certificate state is PENDING.

**Success end condition**: The grant operation for the certificate request is triggered successfully in the Certificate Server and the certificate is installed on the managed node.

**Failure end condition**: The grant operation in the Certificate Server is not initiated, or the grant operation fails, or the certificate is not installed on the node.

**Main success scenario**:
1. The administrator uses the GUI to grant a certificate request.
2. The CSA triggers the grant action for the corresponding request in the Certificate Server.
3. The Certificate Server creates the certificate and sends it to the Certificate Client on the corresponding managed node.
4. The Certificate Client installs the certificate on the managed node and reports success to Certificate Server.
5. The Certificate Server notifies the CSA that the grant operation succeeded.
6. The CSA deletes the mapping information of the node and the certificate request.
7. The CSA updates the state of the certificate request, and shows the result in the GUI.

**Extensions**: None.

**Variations**: If the auto-grant is enabled, and if the certificate request fulfills the auto-grant criteria, the CSA will automatically grant the certificate request.

**Use case 6: Deny Request**

**Description**: This use case allows the user to deny a pending certificate request for a specific node.

**Actors**: Administrator.

**Assumptions**: The pending certificate request has been mapped to a node in the node list in OVO/Windows, and the certificate state is PENDING.

**Success end condition**: The deny operation for the certificate request is triggered successfully in the Certificate Server, and the Certificate Client on the managed node is notified that its request is denied.

**Failure end condition**: The deny operation in the Certificate Server is not initiated, or the Certificate Client is not notified, or the deny operation fails.

**Main success scenario**:
1. The actor uses the GUI to deny a certificate request.
2. The CSA triggers the deny action for the corresponding request in the Certificate Server.
3. The Certificate Server notifies the Certificate Client on the corresponding managed node that its request was denied.
4. The Certificate Client stores the result and reports success to the Certificate Server.
5. The Certificate Server notifies the CSA that the deny operation succeeded.
6. The CSA deletes the mapping information of the node and the certificate request.
7. The CSA updates the state of the certificate request, and shows the result in the GUI.

**Extensions**: None.

**Variations**: None.
Use case 7: Discard Request

**Description:** This use case allows the user to discard a certificate request from the Certificate Server’s list.

**Actors:** Administrator, CSA for action-failed requests.

**Assumptions:** None.

**Success end condition:** The remove operation for the certificate is triggered successfully in the Certificate Server and the request is removed from the Certificate Server’s list of pending requests.

**Failure end condition:** The remove operation in the Certificate Server is not initiated, or the request is not removed from the Certificate Server’s list of pending requests.

**Main success scenario:**
1. The administrator uses the GUI to discard a certificate request.
2. The CSA triggers the remove action for the corresponding request in the Certificate Server.
3. The Certificate Server removes the request from its list of pending requests.
4. The Certificate Server notifies the CSA that the remove operation succeeded.
5. If the certificate state is not UNDEFINED, the CSA deletes the mapping information of the node and the certificate request.
6. The CSA updates the state of the certificate request, and shows the result in the GUI.

**Extensions:** None.

**Variations:** If the grant/deny operation fails, the CSA will automatically discard the corresponding certificate request.

Use case 8: Configure Auto-Grant

**Description:** This use case allows the user to configure auto-grant mechanism.

**Actors:** Administrator.

**Assumptions:** None.

**Success end condition:** The auto-grant is enabled, with different criteria configured; or the auto-grant is disabled.

**Failure end condition:** The auto-grant mechanism cannot be changed.

**Main success scenario:**
1. The actor uses the GUI to trigger and configure auto-grant operation.
2. The actor enables or disables the auto-grant option. If enabled, the actor can also set different criteria for the auto-grant.
3. The configuration is saved.

**Extensions:** The configuration of different criteria for the auto-grant includes: Configure Timeframe, Configure IP Address Range, and Configure Node Group.

**Variations:** None.

Use case 9: Update Trusted Certificates on Node

**Description:** This use case allows the user to update the list of trusted certificates on the node/nodes.

**Actors:** Administrator.

**Assumptions:** None.

**Success end condition:** The list of trusted certificates on the node/nodes is updated.

**Failure end condition:** The update trusted certificates operation fails, and the trusted certificates on the node/nodes are not updated.

**Main success scenario:**
1. The actor uses the GUI to trigger update trusted certificates operation on one node or several nodes.
2. The trusted certificates are transferred to the node/nodes.
3. The update operation succeeds and the result is showed in the GUI.

**Extensions:** None.
**Variations:** Update trusted certificates can also be done via the command line on the managed node.

This chapter describes the design of the CSA system, with regards to the architecture, assumptions and definitions, the detailed process flow, and the use cases. In the next chapter, the actual implementation of the CSA system is presented.
Chapter 5

Implementation

This chapter describes the implementation of the CSA system, which includes the database and its model, system classes and COM interfaces, the graphical user interface, and two supplementary command line tools. The testing of the system, followed by an evaluation, is also presented.

5.1 Implementation

5.1.1 Developing Environment

The developing environment for the CSA was Microsoft Visual Studio 2005, and the programming language was Visual C++.

The CSA is hosted in an existing component of the Security Server, using the same process, in the form of .exe.

5.1.2 Database and Model

Since the CSA is part of the OVO/Windows application system, the database to use for the CSA is decided to be the same one as the whole application system, Microsoft SQL Server Enterprise Edition. This Enterprise Edition includes the complete set of SQL server data management and analysis features. It scales to the performance levels required to support the largest web sites, Enterprise Online Transaction Processing systems, and Data Warehousing systems [36].

For the CSA itself, we need to add one new table to store information about node mapping. Figure 5-1 is a description of the table. OV_RequestNodeMap is the name of the table, ReqID is used to store the ID of the certificate request, NodeID is used to store the ID of the node in the OVO/Windows node list, and MapType is storing the mapping type, with the possible values of Automatic or Manual.

<table>
<thead>
<tr>
<th>OV_RequestNodeMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>ReqID</td>
</tr>
</tbody>
</table>

Figure 5-1 Database table for node-mapping

As discussed in Chapter 4, the OVO/Windows application system is based on the Windows platform, it uses WBEM (Web-Based Enterprise Management) as a common layer of management to pull data, and the data resides in a WBEM repository called CIM (Common Information Model), a conceptual model that describes details required to manage arbitrary objects such as computer systems, network elements, and
applications. Much information in OVO/Windows is stored in the CIM and being accessed via the WMI (Windows Management Instrumentation).

For the CSA, some modifications to the database model are needed:

1. A new field has been added for the class OV_ManagedNode (to store information about managed nodes).
   - CertificateState: An integer is used to represent the certificate state. Possible values are:
     "0" – (UNDEFINED) A new certificate request arrived.
     "1" – (PENDING) A certificate request was mapped and is now pending to be granted.
     "2" – (GRANTED) A certificate request was granted.
     "3" – (DENIED) A certificate request was denied.
     "4" – (FAILED) Certificate grant action failed.
     "5" – (INSTALLED) Certificate was installed on the node.

2. Two existing fields in the class OV_ManagedNode will be used:
   - InstallDate: to store the time when triggering remote agent installation.
   - AgentId: to be used for the OvCoreID.

5.1.3 COM Interfaces and Classes

There are four fundamental classes in the CSA:
- COvOWCSACertRequest (for handling Certificate-Request related functions)
- COvOWCSACertReqCollection (for getting Certificate Requests)
- OvOWCSAStoreAccess (for accessing the information stored in the database and in the model)
- OvOWCSAUpdateHandler (for handling notifications and events)

The CSA also provides a COM interface for the UI clients to access the server side remotely. COvOWCSACertRequest and COvOWCSACertReqCollection are COM objects with the interface of IOvOWCSACertRequest and IOvOWCSACertReqCollection.

Figure 5-2 shows the class diagram including the methods in the COM interfaces.
5.2 User Interface Design

The UI of the CSA will be integrated with the main console of OVO/Windows. This makes the CSA functionality more integrated with other administrative tasks in the OVO/Windows. Also, it provides convenience for the user with all the information in one GUI.

The main console of OVO/Windows is a snap-in to MMC (Microsoft Management Console) [39]. MMC is part of the presentation layer of the Windows management services and serves as a common console framework for server and network management applications known as snap-ins. Snap-ins allow administrators to more effectively manage network resources. The CSA UI will be an extension of the OVO/Windows MMC snap-in, which can fulfill the requirements that we have for the CSA; also, the CSA can make use of some of the existing UI functions from the main console.

Figure 5-3 gives a rough idea of what the UI will look like. A list of pending certificate requests will be shown in the main panel. The columns for displaying the information of certificate requests are: Node, ID (Request ID), CN (OvCoreID), Hostname, IP Address, OS Type, OS Version, System Type, and Received Time. Users can set the options to choose which columns to display. By double-clicking on one request, the user can view the detailed information of that request. Also, by clicking on one request, from the context menu, the user can trigger a Grant/Deny/Discard action, if the request is mapped to a node; if not mapped, the user can trigger the Map Node action to map a certificate request to a node in the OVO/Windows node list.

![Figure 5-3 The CSA User Interface](image)

5.3 Command Line Tools

Two command line tools, ovowcsa and ovowcsacm, are provided that allow triggering all the actions that are available from the GUI also from the command line, which is probably simpler and more familiar for many system administrators.
5.3.1 The Tool ovowcsa

The purpose of the tool ovowcsa is to provide the functionality for listing the pending certificate requests, mapping certificate request with an appropriate node, and also for granting, denying, and discarding specified certificate requests.

Usage:

```
ovowcsa -listpending [-format <format_string>]
-grant <hostname|CertReqID>
-deny <hostname|CertReqID>
-discard <hostname|CertReqID>
-map <hostname|CertReqID>[<=nodelist_hostname|nodeid][-force]
-unmap <hostname|CertReqID>
--help | -h | -?
```

<format_string> = <rhiompts>

- r - Request ID
- h - Hostname
- i - IP Address
- o - OvCoreID
- m - Mapped Hostname
- p - Platform
- t - Time received
- s - Show Header Line

A description of ovowcsa features follows below:

1. -listpending

This switch lists all certificate requests from the Certificate Server's pool of pending certificate requests. By default, the Hostname, RequestID, and Mapped host are output for each certificate request. In addition to the default output the administrator also has the possibility to manually determine the format of the output:

```
-format <format_string>
```

<format_string> = <rhiompts>

This switch can be used only in combination with the -listpending switch. The letters in the format string above mean that the corresponding information will be printed out in a given sequence:

- r - Request ID
- h - Hostname
- i - IP address
- o - OvCoreID
- m - Mapped hostname
- p - Platform
- t - Time received

If "s" (Show header line) is present in the format string, the output will contain the header line with the names of the attributes as the first line of the output. Figure 5-4 gives an example of the ovowcsa output for the command `ovowcsa -listpending -format hipme`, which shows Hostname, IP address, Platform, and Mapped hostname of the pending requests.
2. `-map <hostname|CertReqID>[[=<nodelist_hostname|nodeid>]][-force]

This option maps a certificate request to a node identified by the <nodelist_hostname> from the OVO/Windows node list. This changes the node certificate state of the node in the node list from UNDEFINED to PENDING and adds an entry in the OV_RequestNodeMap table in the database.

Hostname or CertReqID can be given for specifying the pending certificate request. If the certificate request can be determined uniquely by the hostname, then this is used for determining the certificate request, otherwise an error will be given with a message that CertReqID is needed for specifying the certificate request uniquely. nodelist_hostname or nodeid is an optional parameter. If it is not given, then the hostname from the certificate request will be used for the mapping purpose. If the administrator wants to map the certificate request with a node from the node list that has a different hostname than specified in the certificate request and if the node from the node list has an empty OvCoreID, the -force flag has to be used, so that the operation succeeds. The operation will never succeed when the OvCoreIDs from the certificate request and the node in the node list differ.

3. `-grant <hostname|CertReqID>

By using this switch the certificate request with <hostname> or <CertReqID> is granted. If a certificate request can not be determined uniquely by the given <hostname>, an error message will appear, that the <CertReqID> should be used instead. The Certificate Server’s grant-API is called and the node certificate state in the OVO/Windows database will be changed from PENDING to GRANTED.

4. `-deny <hostname|CertReqID>

By using this switch the certificate request with <hostname> or <CertReqID> is denied. If a certificate request can not be determined uniquely by the given <hostname>, an error message will appear, instructing that the <CertReqID> should be used instead. The Certificate Server’s deny-API is called and the node certificate state in the OVO/Windows database is changed from PENDING to UNDEFINED.

5. `-discard <hostname|CertReqID>

By using this switch the certificate request with <hostname> or <CertReqID> is deleted. If a certificate request can not be determined uniquely by the given <hostname>, an error message will appear, instructing that the <CertReqID> should be used instead. The Certificate Server’s delete-API is called and the node certificate state in the OVO/Windows database is changed from PENDING to UNDEFINED.

There are some occasions that the user needs to discard a certificate request from the pool of pending requests. For example, when there are two requests pending for the...
same node. Then, the administrator should map and grant the newer one and discard
the older one.

5.3.2 The Tool ovowcsacm

The tool ovowcsacm provides its functionality from the Certificate Server component,
to be used on the management server, for issuing new node certificates to be
installed manually and for issuing installation keys manually.

Usage:

```bash
ovowcsacm -issue -name <nodename> [ -file <filename> ]
[ -coreid <OvCoreId> ] [ -pass <pass phrase> ]
-geniinstkey [ -file <filename> ] [ -pass <pass phrase> ]
--help | -h | -?
```

A description of ovowcsacm features follows below:

1. `-issue -name <nodename> [ -file <filename> ] [ -coreid <OvCoreId> ] [ -pass
   <pass phrase> ]`

This issues a signed certificate and the corresponding private key for a node and
writes both to the specified file `<filename>`. If the `-file` switch is not specified
ovowcsacm generates the file name in the directory that is readable/writable only for
the system administrator (`<DataDir>/shared/server/certificates`). The filename
is generated in the following format: "<nodename>-<OvCoreID>.p12". (The file is in
the format of PKCS-12 [38], which is the Public-Key Cryptography Standard for
storing private keys and certificates.)

The file can then be moved to a portable medium and transported to the
corresponding node. The `<nodename>` has to be specified as additional information.
The optional `<OvCoreId>` parameter can be used to specify the OvCoreId of the
certificate. If this parameter is empty, a new OvCoreId will be generated for the
certificate. The `<pass phrase>` parameter is for protecting the certificate data that
will be generated. The provided pass phrase is used to calculate an encryption key
that is then used to encrypt the generated certificate data. If the `-pass` option is not
used, a prompt will be displayed where the pass phrase can be entered. Figure 5-5
shows an example of issuing a new certificate for the node `wismar.ovowtest.dom`
with the command `ovowcsacm -issue -name wismar.ovowtest.dom`. A password was
entered at the request of the prompt.

![Figure 5-5 Issuing a new certificate with the command line tool ovowcsacm](image)

ovowcsacm then updates the OVO/Windows node model. The generated OvCoreID is
inserted into the OVO/Windows node model and the node certificate state is set to
GRANTED.

Note: The node with name `<nodename>` has to be already present in the
OVO/Windows node list prior to the `ovowcsacm -issue`, otherwise a new certificate
will not be generated. A certificate can be generated only for a node that is available
in the OVO/Windows node list.
This creates a new installation key, which together with some additional information will be stored in the given file <filename>. If the -file switch is not specified ovowcsacm generates the file in the directory that is readable/writable only for the system administrator (<DataDir>/shared/server/certificates). The filename is generated in the following format: "CertificateIK_<Mgmt Servername>_<uuid>". (uuid stands for universally unique ID, which is a 128-bit integer that is guaranteed to be unique across platforms and applications\(^2\).)

The created file should then be transferred to a node system in a secure way. On the target node it can then be used to initiate a new certificate request that will be encrypted with the installation key. The certificate server will accept exactly one request that is encrypted with this key. This approach offers the advantage that the certificate request (including the private key) is generated on the node system and nevertheless the system can be authenticated by using the installation key. The <pass phrase> parameter is for protecting the installation key that will be generated. The provided pass phrase is used to calculate an encryption key that is then used to encrypt the generated installation key. If the -pass option is not used, a prompt will be displayed where the pass phrase can be entered.

There is also a command line tool on the certificate client side, for initiating a new certificate request to the Certificate Server, adding node certificates and importing the private keys, and also adding certificates to the trusted root certificates. This tool was implemented before this project, so we are not going to discuss it in detail.

5.4 Testing

The testing of the CSA at the moment is focused on functional testing. The test cases are designed according to the use cases and with different inputs; see Figure 5-6.

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>No requests</td>
</tr>
<tr>
<td>Requests</td>
<td>One certificate request</td>
</tr>
<tr>
<td></td>
<td>More certificate requests</td>
</tr>
<tr>
<td></td>
<td>Display in different formats</td>
</tr>
<tr>
<td>Map Request</td>
<td>Automap with same OvCoreID</td>
</tr>
<tr>
<td>to Node</td>
<td>Automap with same hostname</td>
</tr>
<tr>
<td></td>
<td>Manual map with node id which has the same hostname and no OvCoreID</td>
</tr>
<tr>
<td></td>
<td>Manual map with node id, which has the same hostname and same OvCoreID</td>
</tr>
<tr>
<td></td>
<td>Manual map with node id, which has the same hostname but different OvCoreID</td>
</tr>
<tr>
<td></td>
<td>Manual map with node id, which has different hostname</td>
</tr>
<tr>
<td></td>
<td>Manual map with node id, which has different hostname but with force option</td>
</tr>
<tr>
<td></td>
<td>Manual map with node name, which has the same hostname and no OvCoreID</td>
</tr>
<tr>
<td></td>
<td>Manual map with node name, which has the same hostname and same OvCoreID</td>
</tr>
<tr>
<td></td>
<td>Manual map with node name, which has the same hostname but different OvCoreID</td>
</tr>
</tbody>
</table>

\(^2\) The mechanism used to guarantee that uuid is unique is through combination of hardware addresses, time stamps and random seeds. [39]
| Manual map with node name, which has different hostname |
| Manual map with node name, which has different hostname but with force option |
| Un-map Node | Un-map qualified node |
| Un-map node which is in a state other than PENDING |
| Grant Request | Auto-grant certificate request which comes in within the configured time frame |
| | Auto-grant certificate request which comes from a node with IP address in the configured IP address range |
| | Auto-grant certificate request which comes from a node which belongs to one of the configured node groups |
| | Manual grant qualified certificate request |
| | Manual grant certificate request which is in a state other than PENDING |
| Deny Request | Deny qualified certificate request |
| | Deny certificate request which is in a state other than PENDING |
| Discard Request | Discard certificate request |

Figure 5-6 Test Cases

The basic testing measurements for functional testing are:
1) The right database update and CIM model data update
2) The right logic with correct tracing information
3) A consistent interface and reproducible result

Since the CSA is implemented with COM interfaces, it is possible to access and test the functionalities with VB Scripts. So besides the testing from the interface, automatic testing was also conducted using VB scripts.

After functional testing of the system, acceptance testing and user evaluation should be followed. But according to the company practice and the lifecycle of the whole product, we could not do a user evaluation at the moment. Instead, we had an internal evaluation from the user’s perspective. We evaluated the system from the following aspects:

1) How is the data presented to the user? Is all the necessary information there and is it straightforward?
2) How easy are the user operations? For each use case, can users do each within minimal number of steps and minimum interactions?
3) The system should provide necessary warning or error messages when the user performs operations that are not allowed.
4) The use of the system should be straightforward, so that the user can use it without too much reference to the manual.
5) The interface should be consistent and clear.

5.5 Evaluation

After a thorough testing of the system, the result shows that the CSA fulfills all the functionalities as designed. It is reliable enough for all different inputs including invalid ones, and it has a friendly user interface with good performance and stability. An evaluation of the system from different aspects is presented as follows.

5.5.1 Functionality

The CSA as a certificate management system, has the complete functionality to issue certificates, receive certificate requests, evaluate certificate requests, list pending
certificate requests, and grant/deny/discard certificate requests, which are required to set up the PKI environment. Also, the CSA integrates well with the software application system OVO/Windows, to map a certificate request to the node in the OVO/Windows node list, and to set the certificate state of the nodes in the node list.

5.5.2 Usability

The CSA is designed in a way that it should be easy-to-use with a friendly interface. It provides a flexible way to manage certificates according to the user's request for security level. Four types of certificate deployment are offered: manual certificate deployment, manual certificate deployment with one-time installation key, remote manual certificate deployment, and remote automatic certificate deployment. If the user does not have a strong security need, then she can select the weakest certificate deployment method: remote, which is the fastest and most transparent and convenient. If the user has a strong security need in a highly secure environment, she can select manual certificate deployment or manual certificate deployment with one-time installation key. According to the needed security level, the CSA performs more automatic actions to auto grant a certificate request without any human interaction, if a weaker certificate deployment type has been chosen. This way we offer good enough security for all situations without compromise, and at the same time the best convenience and flexibility.

As for a friendly interface, one important feature of the CSA is that it provides a straightforward view of the request as well as the node which sends the request. Detailed information such as hostname, IP address, platform, and arrival time are presented for the user, for easier and better evaluation of the certificate request so that she can decide how to handle it. Currently the User Interface of the CSA is still in the design stage, and the implementation is not finished yet. But with the computer prototype, an internal evaluation of the usability was carried out. The evaluation shows that the CSA as a system is fairly easy to use, with a consistent and informational user interface.

5.5.3 Security

As part of PKI with security as the objective, the CSA is also responsible ensuring that the PKI to be securely set up. The CSA maintains Confidentiality, Integrity, Availability, and Authentication in the following way:

- Confidentiality: Only users with the administrator role assigned are able to access certificate requests. The command line tools are secured with file system permissions. During manual certificate deployment when the certificate or the installation key needs to be transferred to the node system, we should ensure that the courier is fully reliable, it should only be transported by trusted persons, and access to the medium has to be protected to prevent a malicious attacker.

- Integrity: The communication between the CSA and the Certificate Server uses HTTPS. Through the usage of SSL, which uses Message Authentication Codes (MAC) [40], an authentication tag derived by applying an authentication scheme, together with a secret key to a message, data integrity can be assured.

- Availability: Fake or disruptive certificate requests will be denied or discarded.

- Authentication: Through the usage of standard SSL encryption, each communication party provides a certificate (which is signed by a CA and contains the OvCoreID). This is validated during the SSL handshake and it proves the identity of a party.

- Auditing: Each activity of the CSA system will be written to a log file for accountability.
5.5.4 Limitations

As a whole, the CSA system provides full functionalities for managing certificates, with a good usability and security. But there are certain features not included in the scope of this thesis. This section examines the limitations of the CSA system.

1) There is no lifetime management of the created certificates. Instead, the certificates have a very long validity. This constraint is due to the use of an existing security component, the Certificate Server, which gives a long validity when issuing the certificates.

2) There is no GUI for creating certificates or installation keys on the server side, and no GUI for sending certificate requests on the client side; only command line tools are provided.

3) For a multiple-server installation environment, the update of the trusted certificate list on the management server is done manually. Also, at the moment, the synchronization of node certificate state mechanism is not implemented yet.

4) The system has no interfaces to plug-in other PKI solution. At the moment it is implemented for the use in the OVO/Windows product only.

This chapter describes the implementation of the CSA system, and also the test and evaluation. The next chapter is the last chapter of this thesis, which concludes the thesis work and discusses the future work.
Chapter 6

Conclusion

This chapter is a conclusion of the thesis, followed by a discussion of future work.

6.1 Conclusion

Computer security has always been a big and challenging issue in the world of IT industry. PKIs (Public Key Infrastructures) have been developed and used to achieve confidentiality, integrity, authentication, and non-repudiation. As it becomes interoperable, and scalable, more and more products and enterprises start to accept it and incorporate it for a wider use. But this is not easy, especially during the implementation. A successful operation of a PKI relies largely on effective key and certificate management. A properly designed administrative function is a critical success factor.

This thesis focuses on certificate management for secure HTTPS client/server communication, to provide a flexible management for different security requirements and situations, and with a user-friendly interface. A certificate management system CSA has been implemented for a management software product, HP OpenView Operations for Windows.

There are several different ways of managing certificates. A very secure and traditional way is to grant the certificate personally or manually. But it is costly, inconvenient and very time-consuming, especially with large geographically distributed environments. Users will naturally find it very uncomfortable to use. With the automatic remote approach, it is faster and more convenient, but we lose the control on certificate management, thus compromise the security concerns. After comparing these two extremes of certificate deployment, we propose a flexible way according to the user’s request of security level. Four types of certificate deployment are offered: manual certificate deployment, manual certificate deployment with one-time installation key, remote manual certificate deployment, and remote automatic certificate deployment. If the user has a strong security need, she can select the very secure way of deploying certificates; otherwise she can select the faster and more convenient way. This way we offer good enough security for all situations without compromise, and at the same time, the best convenience and flexibility are achieved.

Different from issuing certificates to people, for client/server communication, certificates are issued to computer systems or network nodes. A unique identifier for a computer may not make sense to the user. The CSA provides a straightforward view of certificate requests as well as the node which sends the request. Detailed information such as hostname, IP address, platform, and arrival time is presented to the user, for easier and better evaluation of the certificate request so that she can decide how to handle it.
Testing and evaluation prove that the CSA is a useful system which provides all the functionality for managing certificates, and it is easy to use, and it is secure.

6.2 Future Work

When it comes to the system developed for the purpose of this thesis, there is room for improvement. First of all, the certificate management system CSA that we have developed is implemented specifically for OVO/Windows, a product by Hewlett-Packard. It works solely in the PKI environment used by OVO/Windows, and the system has no interfaces to plug-in other PKI solutions. It would be better and more useful if it could be generalized and independent of one PKI solution.

Also, currently the CSA system has no lifetime management of the created certificates, which means that every certificate has a very long validity. Certificate Revocation Lists (CRLs) [41] can be introduced and with the corresponding management functionality for certificate revocation.

The user interface of the system is not completed with all its expected functionality at the moment. Also it needs to be improved, for example with functions for creating certificates or installation keys on the server side, sending certificate requests on the client side, and updating of the trusted certificate list on the management server. Currently all of these are done from the command line, a friendlier interface ought to be provided to the user.
References


[34] Microsoft Corporation, *Windows Management Instrumentation*,


[38] RSA Laboratories, *PKCS#12: Personal Information Exchange Syntax Standard*,


[40] RSA Laboratories, *What are Message Authentication Codes*,

[41] RSA Laboratories, *What are Certificate Revocation Lists (CRLs)*,
Glossary

CA: Certificate Authority
CIM: Common Information Model
COM: Component Object Model
CRLs: Certificate Revocation Lists
CSA: Certificate Server Adapter
DCE: Distributed Computing Environment
DCOM: Distributed COM
DMI: Desktop Management Interface
DMTF: Distributed Management Task Force
ERP: Enterprise Resource Planning
HTTP: HyperText Transfer Protocol
HTTPS: HTTP over Secure Socket Layer
IDEA: International Data Encryption Algorithm
IMAP: Internet Message Access Protocol
IP: Internet Protocol
ITU: International Telecommunications Union
LDAP: Lightweight Directory Access Protocol
MAC: Message Authentication Codes
MMC: Microsoft Management Console
OLE: Object Linking and Embedding
OVO/Windows: OpenView Operations for Windows
PKCS: Public Key Cryptography Standards
PKI: Public Key Infrastructure
PKIs: Public Key Infrastructures
SNMP: Simple Network Management Protocol
SOAP: Simple Object Access Protocol
SSL: Secure Socket Layer
TCP: Transmission Control Protocol
UUID: Universally Unique ID
WBEM: Web-Based Enterprise Management
WLAN: Wireless LAN
WMI: Windows Management Instrumentation
På svenska

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

In English

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

© [Jing Zhang]