IBM

IBM @server iSeries Wired Network Security

OS/400 V5R1 DCM and Cryptography Enhancements

- Improve SSL performance with the 4758 Cryptographic Coprocessor
- SSL Sockets application examples using the GSKit APIs
- Ensure object integrity with object signing

Thomas Barlen
Barbara Barlocco
Colin Grierson
Vanessa Moffitt
Andreas A. Stadelmann

Redbooks

ibm.com/redbooks

SG24-6168-00

International Technical Support Organization

IBM @server iSeries Wired Network Security: OS/400 V5R1 DCM and Cryptography Enhancements

July 2001

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix H, “Special notices” on page 473.

First Edition (July 2001)

This edition applies to Version 5 Release 1 Modification 0 of Operating System/400 (5722-SS1).

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JLU Building 107-2
3605 Highway 52N
Rochester, Minnesota 55901-7829

Contents

Preface

With the increasing number of customers that conduct business over the Internet or other untrusted networks, there is a rising demand for protecting data traffic. Following several OS/400 releases, the number of TCP/IP applications that support the Secure Sockets Layer (SSL) protocol, as well as the new Transport Layer Security (TLS) protocol, have also increased.

This IBM Redbook focuses on the network security enhancements that are introduced with OS/400 Version 5 Release 1. You will learn how to implement and use the new object-signing capabilities, which allow Business Partners and customer to distribute objects over an untrusted network while assuring their integrity. The redbook also guides you through the redesigned Digital Certificate Manager (DCM) with its new functions, such as Certificate Revocation List processing.

For the e-commerce world where availability, security, and performance are critical to the business, the new 4758 Cryptographic Coprocessor support can help you improve SSL performance and security. One chapter is dedicated to introduce this new support and guide you through the configuration of the cryptographic coprocessor, as well as how to utilize it by DCM.

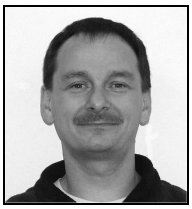
The new Global Secure Toolkit (GSKit) APIs provide better functions and more flexibility when writing SSL Sockets applications. The redbook provides sample code written in ILE RPG to introduce these new APIs.

When working in an environment where products from several vendors are used, you will sooner or later be confronted with questions such as, “This browser version cannot establish a secure session to the server. How can I fix this?”. Most likely, the problems you will face are related to the cryptography support the individual products provide. This is the first publication that provides complete information about the supported encryption and authentication algorithms and key lengths. It also shows you how to control your Web server to accept only certain ciphers for a secure connection by using the new SSL directives.

This redbook targets IBM customers, technical representatives, and Business Partners who are in charge of planning, installing, and implementing network and data security in IBM @server iSeries and AS/400 networks and applications.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Rochester Center.



Thomas Barlen is an IBM Certified Senior I/T Specialist for iSeries and AS/400 systems at the International Technical Support Organization, Rochester Center assigned to the Raleigh Center. He writes extensively and teaches IBM classes worldwide on all areas of iSeries and AS/400 communications and network security. Before joining the ITSO in 1999, he worked in AS/400 software support in IBM Germany. He has over 12 years of experience in AS/400 networking, system management, and security as well as LAN and WAN network design and implementation.

Barbara Barlocco is an I/T specialist in IBM Italy. She has 11 years of experience on the AS/400 platform. Specializing in the communications area, she works in the technical support center in Italy supporting EMEA customers and colleagues. Currently, her focus is on AS/400 Internet technologies and Web server security.

Colin Grierson is a programmer, analyst, and technical consultant for Systems Advisory Services in New Zealand, primarily an outsourcing company using AS/400s. Colin has 22 years of experience in programming and business analysis mainly working on S/38 and AS/400 systems. He holds a degree in mathematics from Auckland University. His areas of expertise include programming, business analysis, and security. He has written applications in RPG/400, Cool 2E (previously Synon/2), and AS/400 CL.

Vanessa Moffitt is an I/T technical specialist working for ASSIST/400 IBM in the United Kingdom. She joined IBM in 1994 and has seven years of experience in the AS/400 system area. She currently provides technical support to AS/400 customers. She has worked in various groups within the AS/400 Support Center in the United Kingdom, including system operations, SEV1/system down, and the hardware group. Now she specializes in communications and networking. Prior to working for IBM, she supported a mainframe system for two years.

Andreas A. Stadelmann is a Dataprotection and Information security consultant and owns a part of KSC AG in Switzerland, a company specializing in enhanced security projects. Andreas has 15 years of experience in programming and business analysis mainly working on AS/400 and mainframe systems. His areas of expertise include security, business analysis and programming. He has written applications in Pascal, RPG/400 and AS/400 CL.

Thanks to the following people for their invaluable contributions to this project:

Tamikia Barrow, Gail Christensen, Christine Johnson, Linda Robinson, Margaret Ticknor, Jeanne Tucker
IBM International Technical Support Organization, Raleigh Center

Jerry Engelbert, Kris Peterson
IBM International Technical Support Organization, Rochester Center

Mike Aho, Mark Bauman, Jim Coon, Jim Fall, Dennis Frett, Rick Hemmer, Terry Hennessy, Timothy D. Mullenbach, Harold Romo, Barb Smith, Rose Sundermeyer, Jay Weeks, Daryl Woker
IBM Rochester Development Lab

Garrett Lanzy, Tom Murphy
IBM Endicott Development Lab

Harry Willemsen
IBM Netherlands

Steve Bireley
Renex Corporation

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Please send us your comments about this or other Redbooks in one of the following ways:

- Fax the evaluation form found in “IBM Redbooks review” on page 487 to the fax number shown on the form.
- Use the online evaluation form found at `ibm.com/redbooks`
- Send your comments in an Internet note to `redbook@us.ibm.com`

Chapter 1. Introduction

This chapter briefly discusses the need for the use of security mechanisms on a public network and introduces the industry-standard techniques for doing this. Most of the issues relating to network security on the AS/400 system are covered in detail in the existing redbook *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659. This book concentrates on the new security features added to the OS/400 in Version 5 Release 1.

Note that when we talk about Secure Socket Layer (SSL) in this book, we also refer to the newer standard Transport Layer Security (TLS).

1.1 Why is network and application security so important?

There is not a single day that you do not hear about new viruses, denial-of-service attacks, stolen credit card numbers, and so on. All these threats are aimed to harm your systems, networks, and applications. Conducting business over an untrusted network, such as the Internet, requires that you protect your and your customers' data from unauthorized access. If you do not, it is very likely that customers will do their future business with someone who can. For different types of attacks, there are different types of security mechanisms to protect your resources. For example, to provide data integrity and confidentiality you can establish secured sessions using the Secure Socket Layer (SSL) protocol or Virtual Private Networks (VPNs) between servers and clients, which ensures that the data while in transit is encrypted and remains unchanged.

Many companies implement a single firewall and consider this device as the one that can protect everybody and everything from threats launched from within the Internet. But most of the attacks are initiated from within the intranet and therefore a typical firewall installation usually cannot provide the required level of protection. What does this mean? It means that you have to implement security in layers. A firewall is certainly a very important link in the chain of security mechanisms and appliances, but you need to implement security in many places throughout your network to achieve a fairly secure environment.

To summarize, there should be no doubt about the need of implementing network, system, and application security. The enhancements of OS/400 Version 5 Release 1 introduce a whole set of new and changed security functions and features allowing you to securely conduct business with other businesses and customers.

1.2 Security goals

The goals and basic concepts of network security are similar to other aspects of security in computer systems. The main difference is that network security often deals with data that is transmitted, parties that are remote and networks that are public and more vulnerable to attacks. Also, the myriad of devices of different characteristics in a network makes network security particularly challenging.

The two central concepts of security are:

Authentication Determine that the users are who they claim to be. The most common technique to authenticate is by user ID and password. Another emerging technique of authenticating communication partners is using digital certificates.

Authorization Permit a user to access resources and perform actions on them. An example of authorization is the permissions on OS/400 objects.

These concepts are necessary to achieve the three primary goals in all types of security:

Confidentiality Only authorized users can view the data. For data that is transmitted through a network there are two ways to achieve this goal: either make sure that only authorized persons can access the network, or encrypt the data.

Integrity Only authorized users can modify the data, and they can only modify it in approved ways. The data is not changed either by accident or maliciously. For data that is transmitted over a network there are two ways to achieve this goal: either make sure that only authorized persons can access the network (not easy to achieve in public networks such as the Internet), or digitally sign the data.

Availability The resources are always available and performing at the expected level. Users can access applications and data at all approved times. The resources have no unexpected downtime as a consequence of an attack.

Network security is also often the first line of defense for securing your host systems. The network is replacing the physical gates and doors to enter your organization. Attackers from outside your organization must break through either your network or your physical security before they can attempt to break your host security.

For more information regarding security concepts and implementation in the iSeries 400 and AS/400 system environment, refer to *AS/400 Internet Security Scenarios: A Practical Approach,* SG24-5954.

1.3 Security protocols and architectures

Because we need to address the security of communications between companies, and most companies work with a broad spectrum of others, we must use industry-standard protocols and products.

Using standard methods is better than using lesser known proprietary methods. Research is constantly being done on the standard methods, and any security weaknesses found are promptly addressed. Therefore, there is a very little chance that there are any unknown and unsolved major flaws in these systems. With proprietary systems, you cannot be so confident.

TCP/IP is the standard communications protocol. This is used both for the Internet and for private networks.

When using TCP/IP communications, traffic is typically secured by using SSL or VPN, as follows:

SSL Secure Sockets Layer is the older and more widely used protocol. The applications communicating have to be written to use SSL. Because the applications do the SSL processing, they can be very flexible. No prior setup is needed before two applications can use SSL to communicate. There are three versions of SSL. SSL V2 is the first version of SSL that was implemented in commercial applications, but nowadays it is not very often used anymore. SSL V3, which also provides client authentication, is the most commonly used protocol to protect network traffic. The Transport Layer Security (TLS) Protocol Version 1 is the first official standard, as described in the Request for Comments (RFC) 2246 and is similar to SSL V3.

VPN Virtual private networks operate on a different layer of the OSI reference model. They can be established by using the Internet Protocol Security (IPSec) framework with the associated protocols on its own or by combining IPSec with the Layer 2 Tunneling Protocol (L2TP). The advantage of building VPNs is that the security is implemented at the IP (IPSec) or the data link layer (L2TP). VPNs are configured as part of the communications setup and therefore applications do not need to be modified to support a secure connection.

Both SSL and VPN can use a number of standard encryption ciphers and authentication methods. The design is such that new ciphers and authentication methods can be added as required. When an SSL or VPN connection is initiated, the parties negotiate to find the best cipher they both support.

User IDs and passwords are still the standard method of authenticating communication partners. However, having thousands of applications with different user and password rules makes it extremely difficult to remember all the used passwords. The result is that people start writing down their passwords, which compromises security policies. The answer to this problem is using digital certificates.

Digital certificates provide another method of authentication, that is of establishing the identity of a party.

- A digital certificate is simply a document that contains information about the identity of the owner and a public key. These data is signed by a trusted agency in such a way that it is almost impossible to forge.
- The public key has a corresponding private key that the certificate owner must store securely. The private key should never leave the owner's system for normal operation.
- The owner of a certificate can publish the certificate for others to use. The public key can then be used to encrypt data for the certificate owner to read. It can also be used to decrypt data sent by the certificate owner. This is normally done to prove the certificate owner was the one that sent the data.

To have a large system that uses digital certificates requires the following:

- Repositories from which applications can obtain the certificates they need
- Recognized authorities who are trusted to issue certificates in a reliable manner
- Defined means of requesting certificates
- Processes that handle certificate revocation

Standards for these functions have been developed, called the Public Key Infrastructure (PKI).

For more information on the use of digital certificates on the AS/400 system, refer to *AS/400 Internet Security: Developing a Digital Certificate Infrastructure*, SG24-5659.

1.4 What's new in OS/400 V5R1?

This section introduces the major enhancements of OS/400 Version 5 Release 1 as they relate to network security.

1.4.1 Digital Certificate Manager (DCM)

DCM is the central utility to administer digital certificates in OS/400. In order for SSL to work, DCM is used to assign digital certificates to applications. The following major changes and enhancements are made to OS/400 V5R1:

- DCM has been reworked to make the user interface more friendly and more intuitive to use. This includes an entire new way of navigating through the various management tasks. Fast path options have been added to simplify common management tasks.
- The passwords used to protect certificate store files are now held in a more secure manner in an internal system object. The stashed password file from previous releases will automatically migrated to the new storage location.
- All applications that communicate over SSL have to be registered in DCM. Prior to V5R1 the user had to use APIs to register or deregister an application. Starting with V5R1, DCM can be used for registering and deregistering user applications through the browser interface.
- New enhancements are introduced to manage SSL application settings, such as whether client authentication is required.
- In case you have a central Certificate Authority (CA) in a corporate network issuing certificates for the company's servers and users, DCM allows you now to add a PKIX location that, when requesting a certificate using DCM, adds an integrated link that points to this central CA.
- Another important function that allows the iSeries server to fully participate in a PKI environment is the Certificate Revocation List (CRL) support. For more information about CRL processing refer to 1.4.4, "Certificate revocation list checking" on page 7.
- New types of certificate stores are added to support OS/400 object signing. For more information about object signing refer to 1.4.3, "Object signing" on page 6.

Refer to Chapter 2, "Digital Certificate Manager" on page 11, for more information about the DCM in V5R1.

1.4.2 4758 PCI Cryptographic Coprocessor for iSeries

The 4758-023 Cryptographic Coprocessor support in V5R1 allows you to improve SSL performance and increase security.

When establishing an SSL or TLS session an SSL handshake is performed. This handshake has a considerable impact on the performance due to public/private key processing. The 4758 PCI Cryptographic Coprocessor for iSeries can now be used to offload the handshake processing work from the main processor to the cryptographic coprocessor. The number of cryptographic coprocessors have been increased from 3 to a maximum of 8, which allows you to share the load between various coprocessors.

The cryptographic coprocessor also allows you to increase system security by storing private keys in the coprocessor or by storing private keys in files encrypted by the master key of the cryptographic coprocessor.

Refer to Chapter 4, “Using hardware cryptography support for SSL/TLS” on page 189, for more information on how to implement and use the cryptographic coprocessor adapter.

1.4.3 Object signing

Objects stored on an iSeries 400 or AS/400 system can now be signed using a specified digital certificate. The signature can be used to verify the object’s integrity and the origination at some later time. This new support is certainly of interest for independent software vendors, business partners and customers who want to ensure that their distributed objects are not changed while in transit.

Only programs, save files, and stream files can be signed.

Starting with V5R1, OS/400 and IBM LPPs will be digitally signed by IBM. Users can verify that programs from IBM have not been altered since they were signed by IBM.

The DCM can be used to sign objects and to verify their signatures. You can also use OS/400 APIs or commands to perform object signing and signature verification tasks. When creating this redbook, we have also created commands to call the APIs.

For a complete description of how to use and deploy object signing, refer to Chapter 3, “Object signing” on page 99.

1.4.4 Certificate revocation list checking

A Certificate Authority (CA) is responsible for maintaining a certificate revocation list (CRL). This list contains information about certificates that have been revoked for various reasons, such as a compromised private key. As part of the DCM enhancements, the customer is now able to check whether a presented client or server certificate has been revoked. To achieve CRL checking, you can now configure DCM to contact the CA's CRL when a certificate is being used.

For more information on CRL processing, refer to Chapter 2, "Digital Certificate Manager" on page 11.

1.4.5 SSL support added to FTP

A function many people were waiting for is now supported in OS/400: SSL support for the FTP server. As a new member of the SSL-enabled applications in OS/400, the SSL support for the FTP server is also activated and managed through the DCM interface. You can configure the server for server authentication only or for both server and client authentication.

Note that currently only the FTP server supports SSL, not the FTP client in OS/400.

By changing the FTP server attributes you can now specify whether you want to allow only SSL connections, only non-SSL connections, or both.

For more information on how to configure and use FTP with SSL, refer to Chapter 5, "Securing OS/400 application traffic with SSL/TLS" on page 283.

1.4.6 SSL client authentication for Telnet server

Prior to OS/400 V5R1, Telnet client authentication was activated by calling a system program. With V5R1 you can enable client authentication through the application settings in DCM.

By changing the Telnet server attributes, you can now specify whether you want to allow only SSL connections, only non-SSL connections, or both.

1.4.7 HTTP servers supports selection of SSL protocol and cipher

Both the original HTTP Server and the HTTP Server powered by Apache now provide the capability of specifying what protocols and cipher suites they accept when establishing a secure connection. In addition you can define whether SSL sessions are cached and when the cached sessions time out.

The new directives are manually configured in the original HTTP Server using the `WRKHTTPCFG` command.

The HTTP Server powered by Apache allows you to define the new directives through the HTTP administration and configuration interface.

Note that, beginning with V5R1, the HTTP ADMIN server instance runs as an Apache server.

For more information about the new directives, refer to Chapter 7, “Ciphers and cryptographic product considerations” on page 373.

1.4.8 SSL support added to Java

You can use SSL to secure communications for the applications that you develop with Developer Kit for Java. Client applications that use IBM Toolbox for Java can also take advantage of SSL. For more information about SSL support in Java, refer to *Securing applications with SSL* found in the iSeries Information Center by clicking **Security->Securing applications with SSL**.

1.4.9 New SSL support for LDAP directory client

A new LDAP directory client has been added to OS/400 V5R1. You can enable SSL for this client to provide a secure connection between the LDAP client and the LDAP server.

1.4.10 New encryption algorithm supported

SSL now supports the Advanced Encryption Standard (AES) algorithm for encryption. AES was developed as a result of a contest for a follow-on standard to DES held by the National Institute for Standards and Technology (NIST). The Rijndael algorithm was selected. This is a block cipher created by Joan Daemen and Vincent Rijmen with variable block length (up to 256 bits) and variable key length (up to 256 bits).

1.4.11 New Global Secure Toolkit (GSKit) APIs

The OS/400 Global Secure Toolkit (GSKit) and OS/400 Secure Sockets Layer (SSL) application programming interfaces (APIs) enable and facilitate secure communications between processes on a network Just as the SSL APIs, GSKit APIs allow you to access SSL and TLS functions from your sockets application program. GSKit APIs provide more options and functionality than the SSL APIs and are the preferred method to secure applications.

We have written sample sockets applications using the nw GSKit APIs. Refer to Chapter 6, “Using SSL in ILE RPG sockets applications” on page 335, for more information on the GSKit APIs and how to use them.

1.4.12 Cryptographic Access Provider products

The Cryptographic Access Provider product 57xx-AC1 40-bit encryption has been withdrawn and therefore is no longer available with OS/400 V5R1.

1.4.13 Miscellaneous security enhancements

Quite a number of implementation changes have been made and new facilities added. The following list provides an overview of these changes and enhancements:

- Various changes have been made to OS/400 that improve the SSL overall performance.
- SSL-enabled asynchronous input/output processing is now supported with sockets applications.
- Serviceability enhancements are added to provide the programmer with better debugging capabilities when writing sockets applications. For more information, refer to *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming**.

Chapter 2. Digital Certificate Manager

This chapter introduces the OS/400 V5R1 Digital Certificate Manager (DCM) function changes and enhancements. DCM is the central tool on the iSeries and AS/400 server for managing digital certificates and secure applications. All system-provided SSL-enabled applications are automatically registered in DCM. A server or client certificate must be assigned to an application to establish a secure connection. You can also operate your own local Certificate Authority (CA). When operating your own CA, you can also issue user certificates for your OS/400 user profiles.

Refer to the following publications for general information about DCM and secure applications:

- For OS/400 releases V4R4 and V4R5:

  AS/400 Internet Security: Developing a Digital Certificate Infrastructure, SG24-5659

- For OS/400 V5R1:
  - *Digital certificate management* found in the iSeries Information Center by clicking **Security->Digital certificate management**
  - *Securing applications with SSL* found in the iSeries Information Center by clicking **Security->Securing applications with SSL**

2.1 Overview of DCM

DCM provides a graphical user interface to manage digital certificates and all related functions, which is becoming more and more important for security implementations in the e-world. With DCM, you can create and manage digital certificates for your users acting as a local CA, or request and process digital certificates from third-party or well-known Certificate Authorities, such as VeriSign or Thawte. Starting with OS/400 V5R1, you can also provide a link to users to submit digital certificate requests to Public Key Infrastructure X.509 (PKIX) Certificate Authorities. You can also manage your secure applications, which includes:

- Adding, changing, and removing application definitions
- Assigning certificates to secure applications
- Defining the CA trust

- Defining whether a certificate is validated by accessing a Certificate Revocation List (CRL)
- Specifying whether client authentication is required

2.1.1 Installation prerequisites

You must have the following prerequisites to use DCM and SSL on the iSeries server:

- 5722-SS1 OS/400 V5R1
- 5722-SS1 option 34 OS/400 - Digital Certificate Manager
- 5722-TC1 TCP/IP Connectivity Utilities
- 5722-DG1 IBM HTTP Server
- Either 5722-AC2 (56-bit) or 5722-AC3 (128-bit) Cryptographic Access Provider

  For more information about Cryptographic Access Provider products and their support, refer to Chapter 7, “Ciphers and cryptographic product considerations” on page 373. Note that in V5R1, the Cryptographic Access Provider with 40-bit encryption (AC1) is not available anymore.

If you want to install the 4758 PCI Cryptographic Coprocessor for iSeries to improve performance for SSL handshake processing, you must also install the 5722-SS1 option 35 Cryptographic Service Provider. For other requirements and details, refer to Chapter 4, “Using hardware cryptography support for SSL/TLS” on page 189.

If you want to use SSL with any Client Access Express or IBM Toolbox for Java component, you have to install one of the 5722-CE2 (56-bit) or 5722-CE3 (128-bit) Client Encryption products. Client Access Express needs one of these products in order to establish a secure connection.

2.1.2 DCM functions and components

OS/400 V5R1 enhances DCM in both functionality and in the graphical user interface (GUI). The GUI has been redesigned and is now more logically structured. It provides various ways of performing the available tasks. This section gives you an overview of the functions available in DCM and some hints on how to find your way through the available navigation paths.

Perform the following steps to start DCM:

1. Start the HTTP server *ADMIN instance.

a. On an OS/400 command line, type the following command to start the server instance:

```
STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
```

Or use the Client Access Operations Navigator to start the server instance.

b. Ensure that the ADMIN server instance is up and running under the QHTTPSVR subsystem.

c. You can use the following command to verify that at least port 2001 is in Listen state:

```
NETSTAT *CNN
```

d. Starting with V5R1, the ADMIN server instance runs as an HTTP Server (powered by Apache) instance. If you enabled SSL in a previous release or you want to use SSL for the ADMIN instance in V5R1, you need to enable it as described in Appendix C, “Enabling SSL for the ADMIN server instance” on page 415.

2. Start a Web browser.

In this chapter, we used Microsoft Internet Explorer 5.01 to run our processes.

a. To avoid browser caching problems when using DCM, you should change the following setting in the Internet Explorer (IE) configuration. Click **Tools -> Internet Options -> Settings** and select the **Every visit to the page** box.

b. We also recommend that when working in a local area network, you should bypass the proxy server for accessing your iSeries or AS/400 server. On the IE action bar, click **Tools -> Internet Options -> Connections -> LAN Settings** and check the **Bypass proxy server for local addresses** box.

3. Enter the URL:

```
http://servername:2001
```

The port number 2001 is used to access the HTTP *ADMIN server instance with the HTTP protocol. The URL value *servername* represents the host name or IP address. If SSL is already enabled for the ADMIN server instance, you can also start a secure connection by using the URL:

```
https://servername:2010
```

4. Sign on to the AS/400 Tasks page.

To have full operability in DCM, you need to sign on with a user profile with `*ALLOBJ` and `*SECADM` special authorities. Ordinary users can only manage

their user certificate, view the object signatures for those objects they are authorized to, or sign objects with object signing applications they are authorized to use.

5. Click **Digital Certificate Manager**.

   If you do not see this icon on the AS/400 tasks page, you probably have not installed the OS/400 option 34 (Digital Certificate Manager).

> **Note**
>
> You must have already installed one of the cryptographic access provider products on your system before using the Digital Certificate Manager (DCM) functions.

Figure 1 shows what appears when you select the Digital Certificate Manager from the AS/400 Tasks page in a brand new environment.

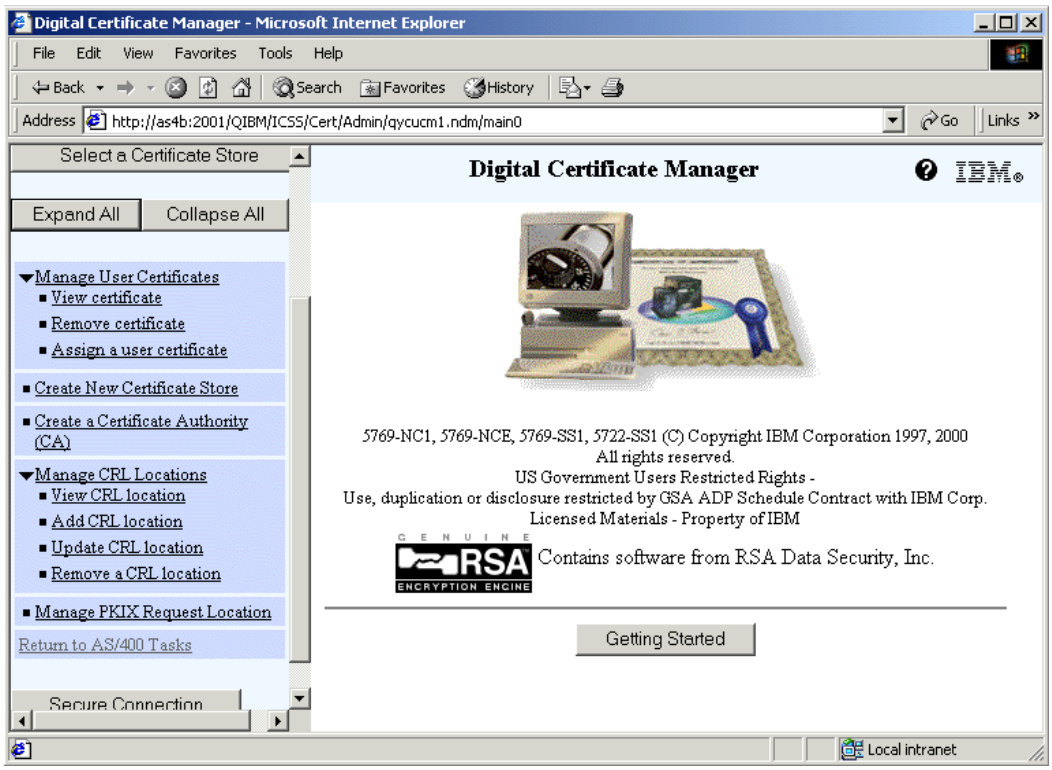


Figure 1. Main DCM window

Click **Expand All** to obtain the same view. By default, option menus on the DCM page are collapsed.

The help system provides several ways for you to access more information about DCM functions. For a quick guide to getting started with this new version of DCM, click **Getting Started** in the right-hand pane or obtain help text for each function by clicking the question (**?**) mark. If any item of the left-hand (navigation) pane is an option menu that contains more than one task, an arrow appears to the left of it. Click the arrow and an expanded list of tasks will appears. But if you click the category link, you can also obtain a brief description in the right pane of the available tasks, so that you may better choose which task to perform.

To establish an SSL session with the *ADMIN server instance, you can click **Secure Connection** and a second browser window appears that initiates a secure connection. This button does not automatically enable the instance for an SSL environment. For this to work, when you have not already done, refer to Appendix C, “Enabling SSL for the ADMIN server instance” on page 415.

DCM gives you different ways to perform a function, but the functions allowed, that appear in the navigation pane, depend on which certificate store you have selected. To switch certificate stores, click **Select a Certificate Store** in the navigation pane. You will access different types of digital certificates, and relative certificate and application management tasks. This is a big change in the navigation of the DCM that better reflects what actually happens on the system. To learn more about each certificate store and its functions, refer to 2.1.3, “Certificate stores” on page 16. But generally with the DCM utility, you can perform the following tasks:

- **Act as a local Certificate Authority**: DCM allows you to create and manage your own local private CA, renew it if the validity period has expired or if you want to change some contents of the CA certificate or define, and change the policy data to which your local CA refers when issuing certificates. You can then use the private CA to dynamically issue digital certificates for your or other AS/400 applications and for users on your intranet or extranet.
- **Manage certificates**: DCM allows you to request, manage, import, export, etc., different types of digital certificates. There are several types of digital certificates depending on the type of usage. You can have CA certificates, server, client, or user certificates, object signing certificates, or signature verification certificates.
- **Manage application definitions**: You can now use DCM to create and update application definitions and manage the certificates that they use.

This allows you to easily use DCM to manage certificates for applications that you write or applications you obtain from other sources that need secure functions. You can define the type of application (server, client, object signing). Depending on the type of application, you can specify whether it performs CRL processing, requires client authentication, or requires a CA trust list.

- **Object signing and signature verification**: You can now use DCM to create and manage certificates that you can use to digitally sign objects to ensure their integrity and provide proof of origination for objects. You can also create and manage the corresponding signature verification certificates that you or others can use to authenticate the signature on a signed object to ensure that the data in the object is unchanged to verify proof of the object's origination. In addition, DCM or corresponding APIs can be used to sign an object, verify the signature on an object, and display signatures on a signed object.
- **Manage Certificate Revocation List (CRL) locations**: DCM now supports using CRLs to provide a stronger certificate and application validation process. You can use DCM to define the location where a specific CRL resides on a Directory Services (LDAP) server so that DCM and other applications that perform CRL processing can verify that a specific certificate has not been revoked.
- **Manage PKIX request location**: Another function that is available with DCM in V5R1 is to obtain and manage certificates from CAs that support the Public Key Infrastructure X.509 (PKIX) standards by defining the location of the CA that you want to use. You can then use DCM to access the URL for the PKIX CA directly to obtain a certificate from the CA.

2.1.3 Certificate stores

A certificate store is a special key database file that DCM uses to store digital certificates and their associated private keys. DCM allows you to create and manage several types of certificate stores. Certificate stores are classified based on the types of certificates that they contain. The management tasks that you can perform for each certificate store vary based on the type of certificate that the certificate store contains.

For example, you have to be in the *SYSTEM certificate store if you want to assign a digital certificate to an application for SSL purpose. But if you want to use that digital certificate to sign an object, you have to export the certificate into the *OBJECTSIGNING certificate store. Then you have to be in the *OBJECTSIGNING certificate store to sign the object.

It is always possible on the navigation pane to click **Select a Certificate Store** to change a certificate store and, therefore, related functions. Figure 2 on page 18 shows the certificate stores that have already been created for you to select. There are some operations that you may perform regardless of the certificate store that is selected. These are:

- **Install Local CA Certificate on Your PC**: This option allows you to install the CA certificate in your browser or to copy and paste the CA certificate into a file on your PC. Which method you choose to install the CA certificate depends on the type of applications that you use to establish secure communications. Such applications include most standard browsers and many non-browser applications, such as Client Access Express and IBM Personal Communications.

  You must install the CA certificate so that these applications can authenticate certificates that the CA issues. These applications can then use SSL to negotiate secure communications sessions with servers that present a certificate that this CA issued.

- **Manage CRL Locations**: This option allows you to choose the type of management task that you want to perform for Certificate Revocation List (CRL) locations. For more information, see 2.5, “Exploiting the Certificate Revocation List support” on page 82.

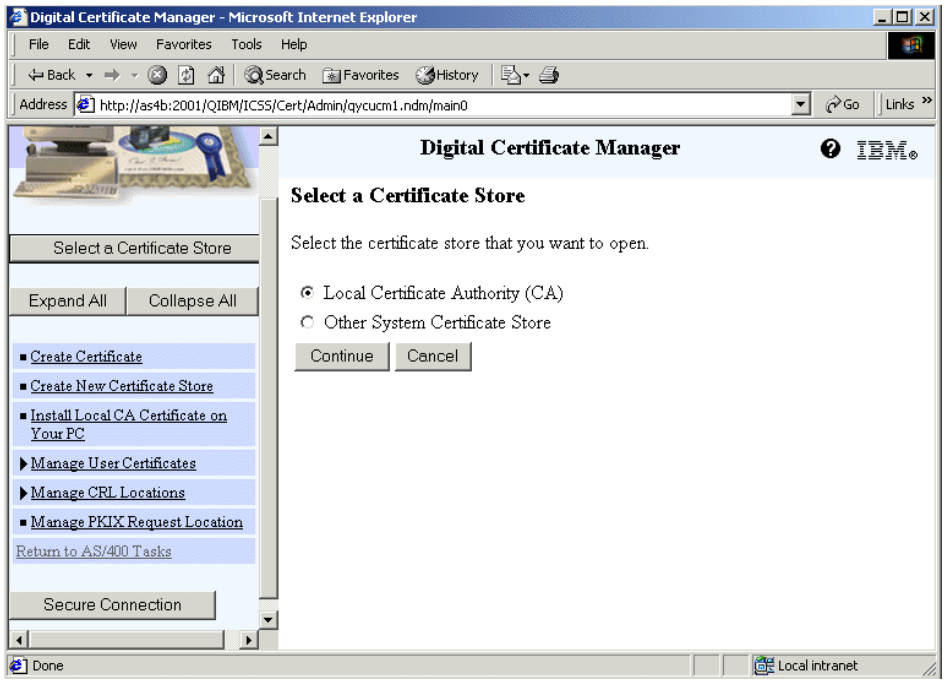


Figure 2. Select a Certificate Store task

- **Manage PKIX Request Location**: This option allows you to specify the fully qualified URL of a Public Key Infrastructure for X.509 (PKIX) Certificate Authority. Setting a URL for a PKIX CA configures DCM to provide a PKIX CA as an option for obtaining signed certificates. A PKIX CA provides a means of using X.509 digital certificates within a public key infrastructure that conforms to the latest Internet standards as outlined in Request For Comments (RFC) 2560. Lotus Domino provides a PKIX CA, for example.
- **Create a New Certificate Store**: If you need to manage the server digital certificate and you do not see *SYSTEM among the selectable certificate stores, you need to create it. Figure 3 shows you which certificate store you can still create.

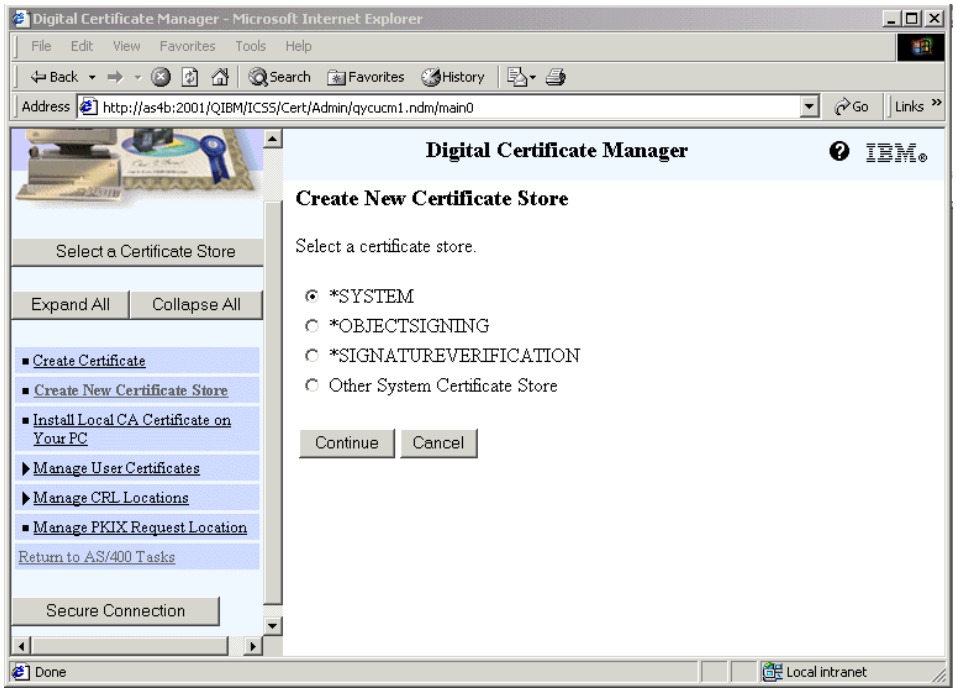


Figure 3. Create a New Certificate Store task

To create a new certificate store, follow these steps:

1. Click **Create New Certificate Store**.

   The window shown in Figure 3 appears.

2. Click **Continue** to confirm the certificate store you have selected to create.

   As shown in Figure 4 on page 20, the new certificate store will contain the default list of CAs, the well-known public CA, and the local CA if it exists. You can choose whether to create an object signing certificate. This is not the only opportunity to do this. Refer to 2.3, “Managing certificates” on page 53, for more information.

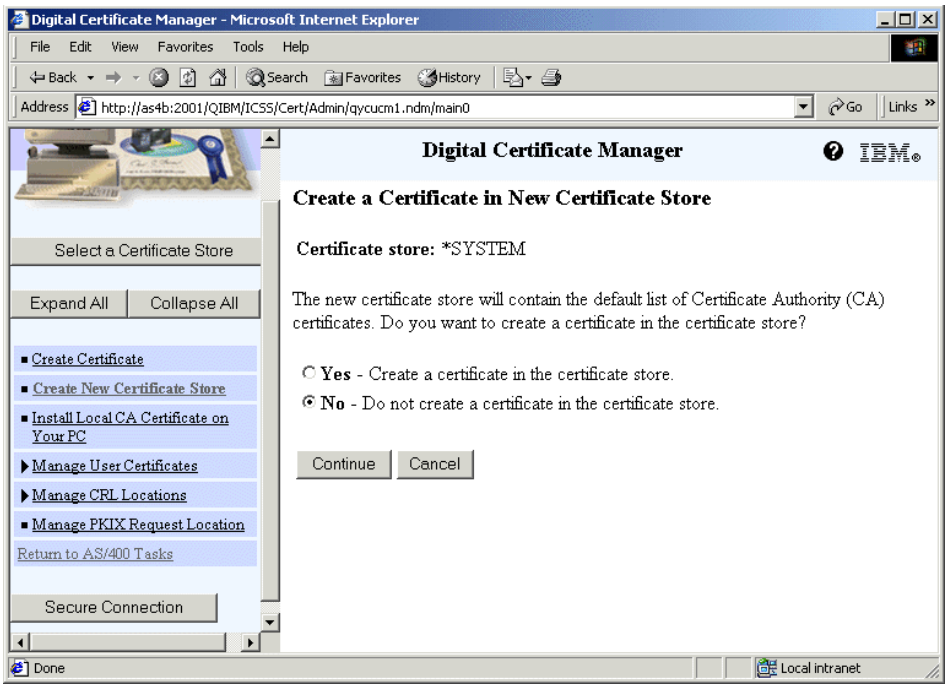


Figure 4. Create a Certificate in New Certificate Store task

3. Click **Continue** to confirm that at this moment you want create just the certificate store.

   The window shown in Figure 5 asks you to enter the password of the certificate store. Note that passwords are *case sensitive*.

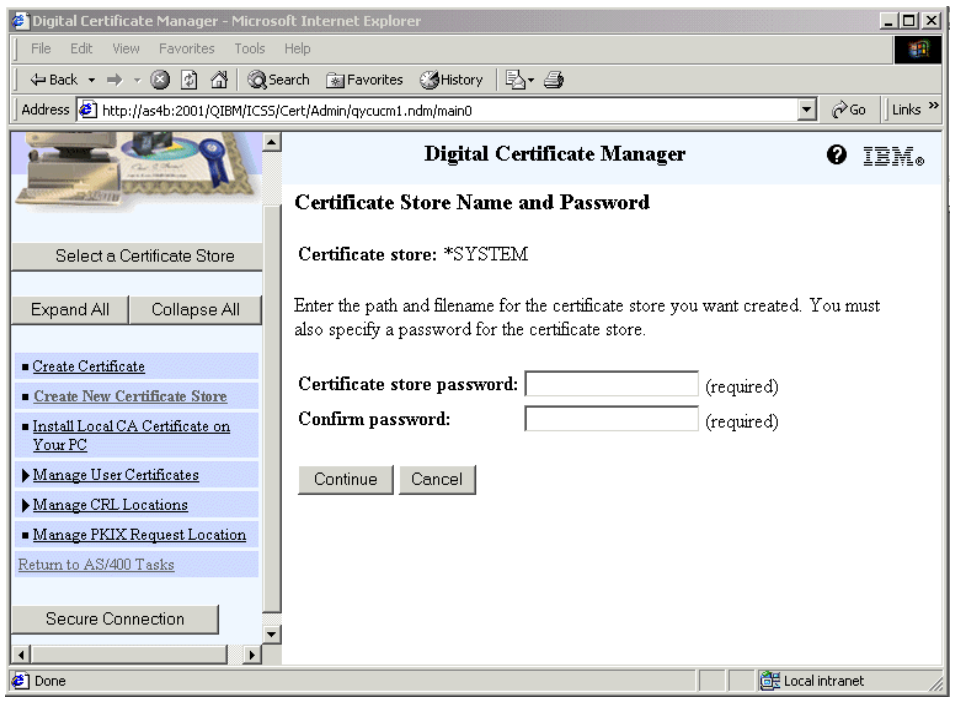


Figure 5. Certificate Store Name and Password task

4. Enter the password, and click **Continue** to proceed.

   In the next window, you see a message that confirms the result of the operation.

There are different certificate stores based on their particular purpose. You have to select one to perform the operations related to their function. DCM provides the predefined certificate stores that are explained in the following sections.

2.1.3.1 Local Certificate Authority (CA)

DCM uses this certificate store to store the local CA certificate and its private key if you create a local CA. You can use the certificate in this certificate store to sign certificates that you use the local CA to issue. However, when the local CA exists, its CA certificates are added in the list of well-known CAs that are available in all the certificate stores for authentication purposes. Applications use CA certificates to verify the origination of certificates that they must validate as part of the SSL negotiation to grant authorization to resources.

When you create a local Certificate Authority store, DCM uses a fixed location to store the keys. After you create a Certificate Authority, you will see the following objects in the IFS:

- **/QIBM/UserData/ICSS/Cert/CertAuth**: Directory
- **CA.TXT**: CA certificate and public key
- **DEFAULT.KDB**: CA key database file
- **DEFAULT.POL**: CA policy file
- **DEFAULT.RDB**: Certificate request file
- **/QIBM/UserData/ICSS/Cert/Download/CertAuth**: Directory
- **CA.CACRT**: CA certificate available for distribution to clients

Note that there is no DEFAULT.STH file. As we mentioned earlier, the password is now stored in a system index.

These directories and objects must be protected from unauthorized access. Therefore, both the /QIBM/UserData/ICSS/Cert/CertAuth directory and all the objects that belong to it have public authority *EXCLUDE. The exception is the /QIBM/UserData/ICSS/Cert/Download/ subdirectory, which has *RX as the public authority for it and its objects to allow downloading of the certificates.

When you select **Local Certificate Authority (CA)** in the Select a Certificate Store task, you see the window shown in Figure 6, which asks you to enter the password.

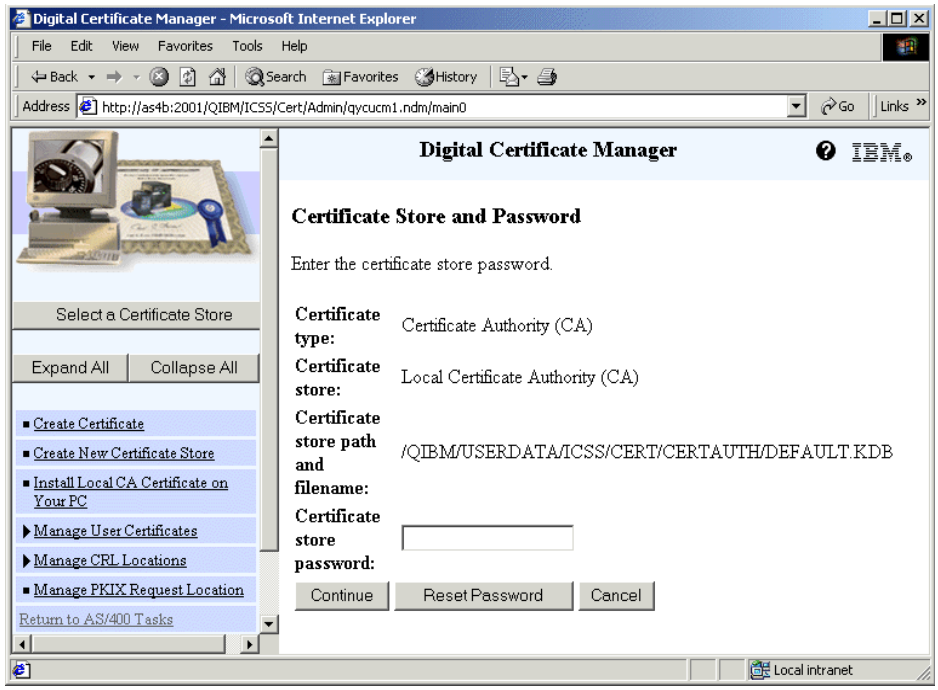


Figure 6. Enter the selected certificate store password

V5R1 now allows you to reset the password of the certificate store using DCM. The same function is provided in V4R5 by calling a program to reset a certificate store password to DEFAULT. This program can be invoked from the command line with the command:

```
CALL QICSS/QYUTRSETPW PARM('/AAA/BBB.KDB)
```

Here `AAA` is the certificate store path name, and `BBB` is the certificate store name. This program requires the user to have *SECADM and *ALLOBJ special authority.

Enter the password and click **Continue**. Then the selected certificate store appears as shown in Figure 8 on page 24. If you press the Enter key on your keyboard, instead of clicking Continue with your mouse, you see the error shown in Figure 7. Although this is a problem for V5R1, it will be corrected in a future release.

```
NET.DATA Error: No HTML(NONE) section in
object /QIBM/ProdData/HTTP/Protect/ICSS/Cert/Macro/qycukdb.ndm.
```

Figure 7. Message error when you press the Enter key

Also the navigation pane is refreshed to show the task list for this certificate store. It also shows the task list for the local Certificate Authority certificate store, as shown in Figure 8 on page 24. The tasks allow you to:

- **Create a Certificate**: The certificate could be a server or client certificate for another AS/400 to use for secure communication sessions, an object signing certificate for another AS/400 to use to sign objects, or a user certificate for the user profile. Note that you cannot create a server or client certificate for this AS/400. This option is available for the *SYSTEM and Other System Certificate stores only. Similarly, you cannot create an object signing certificate for this AS/400. This option is available for the *OBJECTSIGNING certificate store or Other System Certificate store only.

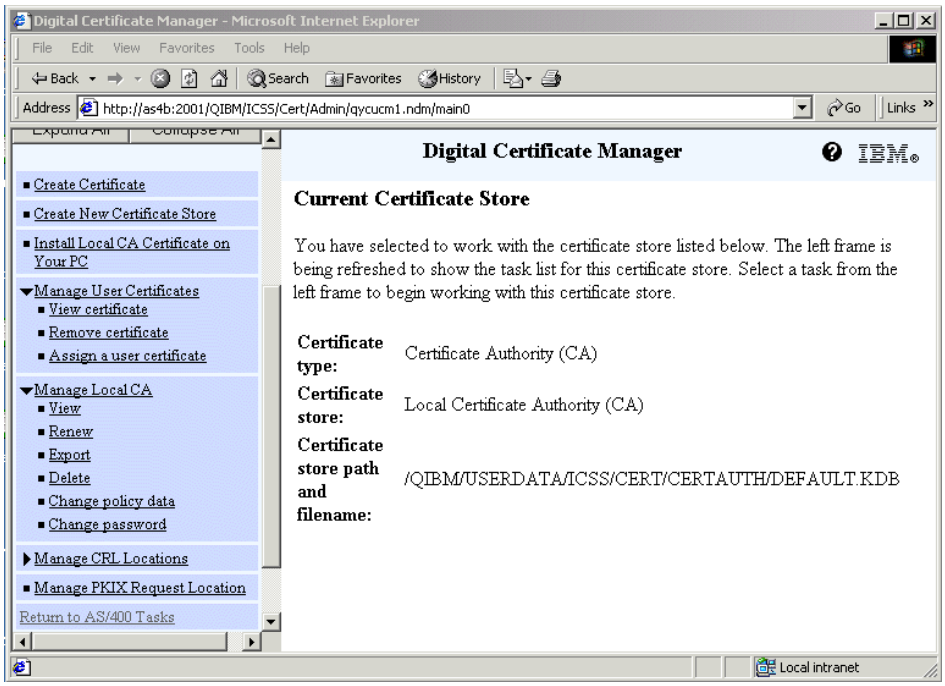


Figure 8. Current Certificate Store message for local Certificate Authority

- **Manage Local CA**: This option allows you to:

- View the distinguished name and other information for the local CA
- Renew your existing local CA certificate
- Export a copy of the local CA certificate to a file or to another certificate store on this system
- Delete the local CA
- Change the policy data for the local CA
- Change the password that protects access to the local CA
- Update the device assignment in order to make changes to the current cryptographic device assignment for the local CA private key

This option is available only if a cryptographic coprocessor is installed and active on this system. For details, refer to Chapter 4, “Using hardware cryptography support for SSL/TLS” on page 189.

2.1.3.2 *SYSTEM

DCM provides this certificate store for managing server or client certificates that applications use to participate in Secure Sockets Layer (SSL) communications sessions. IBM iSeries applications (and many other software developers' applications) are written to use certificates in the *SYSTEM certificate store only.

When you create a *SYSTEM store, the DCM uses a fixed location to store the keys. After you create it, you will see the following objects in the IFS:

- **/QIBM/UserData/ICSS/Cert/Server**: Directory
- **DEFAULT.KDB**: Digital certificates database file
- **DEFAULT.RDB**: Certificate request file

The DEFAULT.KDB contains both the CA certificates and eventually the server or client certificates. As the local CA certificate store, the *SYSTEM certificate store also has to be protected from unauthorized access. Therefore, both the /QIBM/UserData/ICSS/Cert/Server directory and all the objects that belong to it have *EXCLUDE public authority.

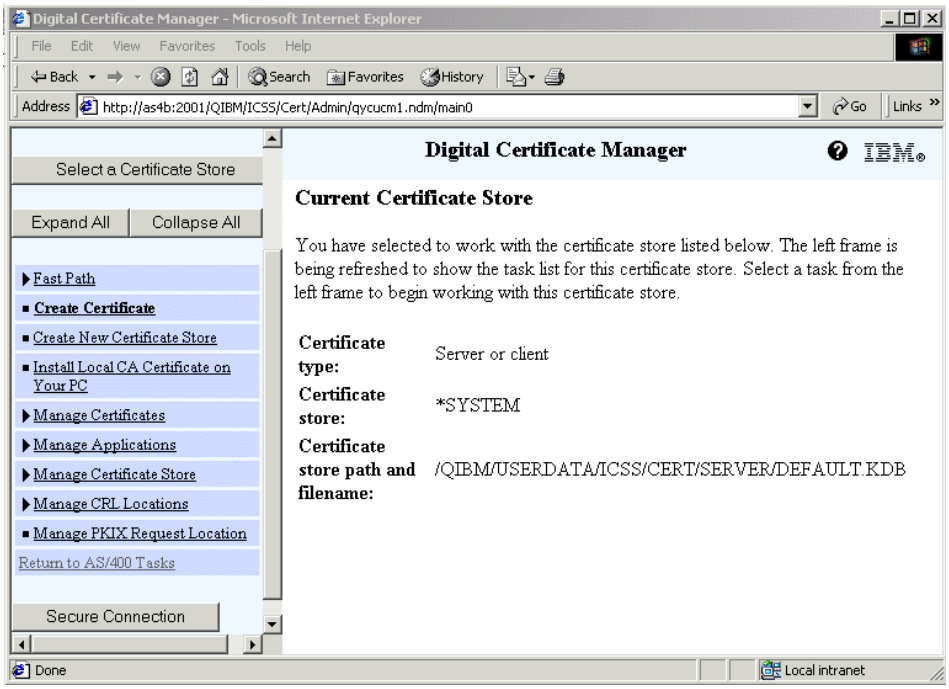


*Figure 9. Current Certificate Store message for *SYSTEM*

As you can see in Figure 9, the navigation pane now shows the possible task list for this store. With the exception of the Fast Path category, each task in the navigation frame is a guided task that takes you through a series of steps to complete the task quickly and easily.

The Fast Path category provides a cluster of certificate and application management functions that allow experienced DCM users to quickly access a variety of related tasks from a central set of pages. Click **Fast Path** to select a category of objects with which to work.

When you select a category, a Fast Path page displays you to work with a cluster of management functions for the selected objects. Fast Path pages are not available when a certificate store is not selected or for the local Certificate Authority (CA) store. Each centralized Fast Path function is also available as a guided task within the main navigation pane. In the *SYSTEM certificate store, you have the following options, which are different from the local CA certificate store:

- **Create a Certificate**: The certificate could be a server or client certificate that is eventually issued by your local CA or a well-known CA, to enable

secure communication sessions by using the SSL (refer to 2.2, “Creating a server or client certificate” on page 33), a server or client certificate for another AS/400 to use for secure communication sessions, or a user certificate for the user profile. By selecting one of the last two options, you will use your local Certificate Authority (CA) to issue that type of certificate. Therefore, those options do not display if you do not have a local CA on the system.

- **Manage Certificates**: This option could be used to choose the type of management task that you want to perform for certificates in the current certificate store. The types of tasks that you can perform depend on the certificate store with which you are working. This page is not available when you are not in a certificate store or when you are in the local Certificate Authority store. For details and examples, refer to 2.3, “Managing certificates” on page 53.
- **Manage Applications**: This option could be used to choose the type of management task that you want to perform for application definitions in the current certificate store. The types of tasks that you can perform depend on the certificate store with which you are working. Manage Applications tasks are available in the *SYSTEM and *OBJECTSIGNING certificate stores only. For details and examples, refer to 2.4, “Managing applications” on page 71.
- **Manage Certificate Store**: This option could be used when you want to manage the current certificate store. This allows you to specify the selected certificate as the default certificate for the current certificate store, to change the password for the current certificate store, or to delete the current certificate store and all certificates and associated private keys in the store.

2.1.3.3 *OBJECTSIGNING

DCM provides this certificate store for managing certificates that you use to digitally sign objects. Also, the tasks in this certificate store allow you to create digital signatures on objects.

When you create an *OBJECTSIGNING store, the DCM uses a fixed location to store the keys. After you create it, you will see in the IFS the following objects:

- **/QIBM/UserData/ICSS/Cert/Signing**: Directory
- **SGNOBJ.KDB**: Digital certificates database file
- **SGNOBJ.RDB**: Certificate request file

The /QIBM/UserData/ICSS/Cert/Signing directory and all the objects that belong to it have public authority *EXCLUDE.

For more information, refer to Chapter 3, “Object signing” on page 99.

2.1.3.4 *SIGNATUREVERIFICATION

DCM provides this certificate store for managing certificates that you use to verify the digital signature on objects.

When you create a *SIGNATUREVERIFICATION store, the DCM uses a fixed location to store the keys. After you create it, you will see the following objects in the IFS:

- **/QIBM/UserData/ICSS/Cert/Signing**: Directory
- **VFYOBJ.KDB**: Digital certificates database file
- **VFYOBJ.RDB**: Certificate request file

In this case, the /QIBM/UserData/ICSS/Cert/Signing directory and all the objects that belong to it have *EXCLUDE public authority. For more information, refer to Chapter 3, “Object signing” on page 99.

2.1.3.5 Other System Certificate Stores

This certificate store option provides an alternate storage location for server or client certificates that you use for SSL sessions. Other System Certificate Stores are user-defined secondary certificate stores for SSL certificates. Mainly, the opportunity to select a non-standard certificate store exists to provide compatibility with pre-V4R4 implementations. The Other System Certificate Store task allows you to manage certificates for applications that you or others write that use the SSL_Init or gsk_environment_init() APIs to programmatically access and use a certificate to establish an SSL session. This API allows an application to use the default certificate for a certificate store rather than a certificate that you specifically identify.

Most commonly, you use this certificate store when migrating certificates from a prior release of DCM, or to create a special subset of certificates for SSL use. For example, in a user-defined directory, a certificate store that has been restored from another system can be used to import digital certificates into the *SYSTEM certificate store at another time. There is no predefined location for these key database files. Therefore, each time you select the Other System Certificate Store, you have to specify the absolute path and the key database file, which could have the extension .KDB or .KYR depending upon whether it was created before or after the V4R4 release. You also have to take care of the security of these objects. The key database file (.KDB or .KYR), the Certificate request file (.RDB), and the directory that contains them must be protected from unauthorized access defining *EXCLUDE as public authority.

Figure 10 shows that you only have access to the Manage Certificates tasks, and not the Manage Applications tasks.

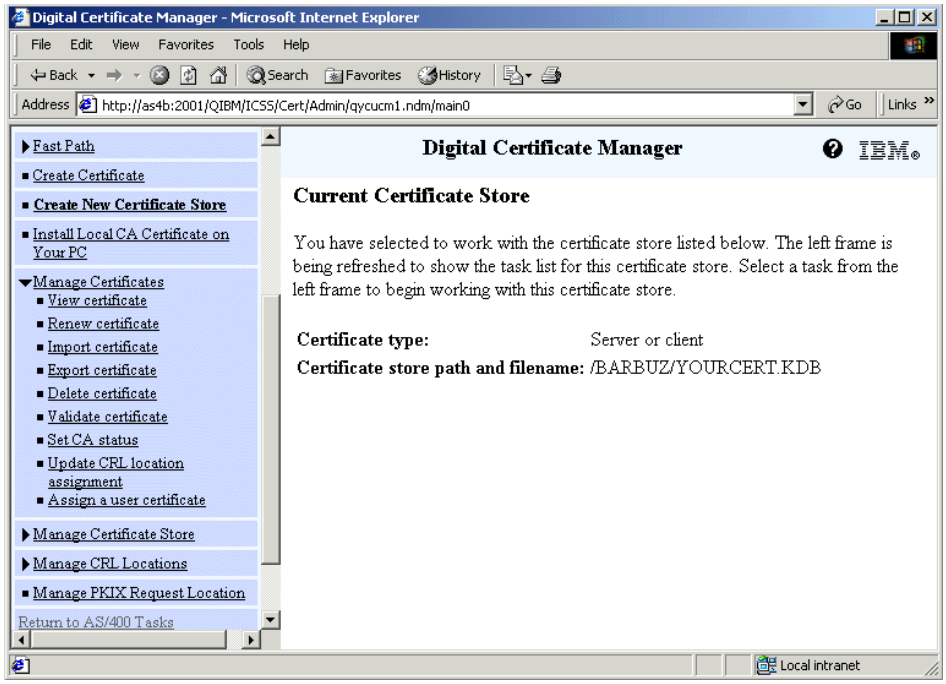


Figure 10. Current Certificate Store information window

2.1.4 Certificate types

A digital certificate is an electronic means of identifying a user that describes an individual, organization, or system. A digital certificate binds you or your organization identity to a pair of electronic keys, the public and the private one, that gives you the ability to encrypt data using a key that only individuals with the appropriate other key can decrypt. More and more, they are being used to:

- Configure applications to use the Secure Sockets Layer (SSL) for secure communications sessions.
- Authenticate users to eventually authorize sessions between server applications and users.
- Configure virtual private network (VPN) sessions. Starting with the V5R1, you can use digital certificate as a means of authenticating a communications partner when establishing a VPN connection.

- Sign objects and signature verification, starting with V5R1, to ensure data integrity as well as the source of the object's origin.

Based on how the certificate is used, there are several classifications of digital certificates.

2.1.4.1 Certificate Authority (CA) certificates

The Certificate Authority is a trusted part that issues digital certificates to users and server or client applications. The trust in the CA is the foundation of the trust in the certificate as a valid credential.

But who validates the CA? A Certificate Authority certificate is a digital credential that provides information about the identity of the CA that owns the certificate. The CA's certificate contains identifying information about the CA and its public key. Others can use the CA certificate's public key to verify the authenticity of the certificates that the CA issues and signs. A CA certificate can be signed by another CA, such as VeriSign, or it can be self-signed if it is an independent entity like the CA that you created with DCM. An example of a self-signed CA is the local CA created by DCM on the AS/400 system.

While the trust of a local CA is based on the organization, for the public well-known ones there is a hierarchy to respect. Some CAs, such as VeriSign or Thawte, are root CAs. They can issue a server digital certificate directly or decide to sign other CA certificates. In that case, it is called an *intermediate CA*, which can issue other digital certificates. Figure 11 shows this possible scenario.

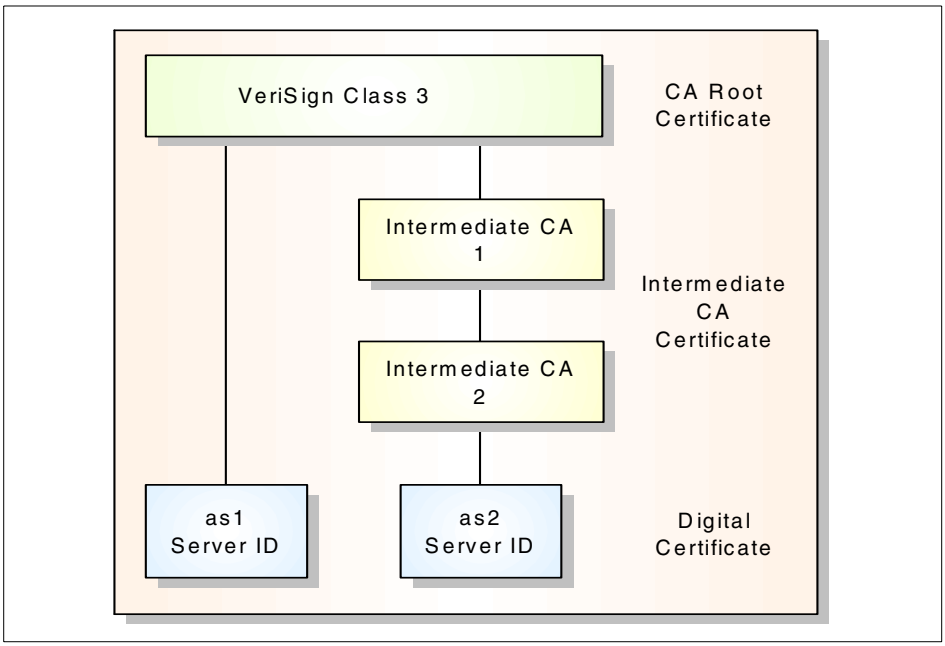


Figure 11. CA structure

To accept a certificate issued by one of the intermediate CAs, you need to import and trust all the CAs in the chain, from the root to the issuer of the digital certificate. That means, according to Figure 11, you can directly import the “as1” server certificate ID because it is signed by a root CA. But, to receive the “as3” server ID, you have to receive the “intermediate CA 1” and the “intermediate CA 2” in sequence.

You will see an actual example of an intermediate CA when you request from VeriSign a Global Server Certificate. In that case, you have to receive the CA certificate of the issuer (VeriSign Trust Network is the common name) before you receive your server certificate. Otherwise, an error such as `Issuer's key not found or Not Trust root` can occur. This will impact both your server (AS/400 HTTP server) and your clients (browsers), so you must consider it during the planning of your SSL implementation.

2.1.4.2 Server or client certificates

A server or client certificate is a digital credential that identifies the server or client application that uses the certificate for secure communications. Server or client certificates contain identifying information about the organization that owns the application, such as the system's distinguished name. The

certificate also contains the system's public key. A server must have a digital certificate to use the Secure Sockets Layer (SSL) for secure communications.

Applications that support digital certificates can examine a server's certificate to verify the identity of the server when the client accesses the server. The application can then use the authentication of the certificate as the basis for initiating an SSL-encrypted session between the client and the server. These certificates are managed in the *SYSTEM certificate store.

In V5R1 when you use local CA to issued a server or client certificate, you can provide additional distinguished name information as the Version 4 IP address, the fully qualified domain name, or the e-mail address. These certificate extensions, called *Subject Alternative Name*, are not necessary for Secure Sockets Layer (SSL), but are recommended for virtual private networks (VPN).

2.1.4.3 Object signing certificates

An object signing certificate is a certificate that you use to digitally sign an object to ensure the integrity of both the object itself and the origination or ownership of the object. When you use an object signing certificate's private key to sign an object, the receiver of the object must have access to a copy of the corresponding signature verification certificate to properly authenticate the object signature. You can manage these certificates by selecting the *OBJECTSIGNING certificate store. For more information, refer to Chapter 3, “Object signing” on page 99.

2.1.4.4 Signature verification certificates

A signature verification certificate is a copy of an object signing certificate without that certificate's private key. You use the signature verification certificate's public key to authenticate the digital signature created with an object signing certificate on an object. You can manage these certificates by selecting the *SIGNATUREVERIFICATION certificate store. For more information, refer to Chapter 3, “Object signing” on page 99.

2.1.4.5 User certificates

A user certificate is a digital credential that validates the identity of the client or user that owns the certificate. Many applications now provide support that allows you to use certificates to authenticate users to resources instead of user names and passwords.

Digital Certificate Manager (DCM) automatically associates user certificates that your local CA issues with the user's OS/400 user profile. You can also use DCM to associate user certificates that other Certificate Authorities issue

with the user's system user profile. You can manage the user certificate without having done any certificate store selection. The only function introduced in V5R1 is that you can now also enter the fully qualified e-mail address when requesting it from your local CA.

For more information about user certificate, refer to the proper section of *Digital certificate management* found in the iSeries Information Center by clicking **Security->Digital certificate management**. You can also refer to *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659.

When you use Digital Certificate Manager (DCM) to manage your certificates, DCM organizes them by these classifications and places them in a certificate store. The associated private keys for the local CA, server, and client certificates can be stored in a certificate store or, if present, stored in the 4758 PCI Cryptographic Coprocessor for iSeries or a key store file protected by the Cryptographic Coprocessor's master key.

2.2 Creating a server or client certificate

This section covers the steps to request a digital certificate for securing your applications. To obtain a digital certificate, you have to create a "request" first. Figure 12 on page 34 shows the possible choices. You can select **Create Certificate** and then **Server or client certificate** in the *SYSTEM or in other user certificate stores. You will see the local CA option if it is created and the PKIX location if it is defined.

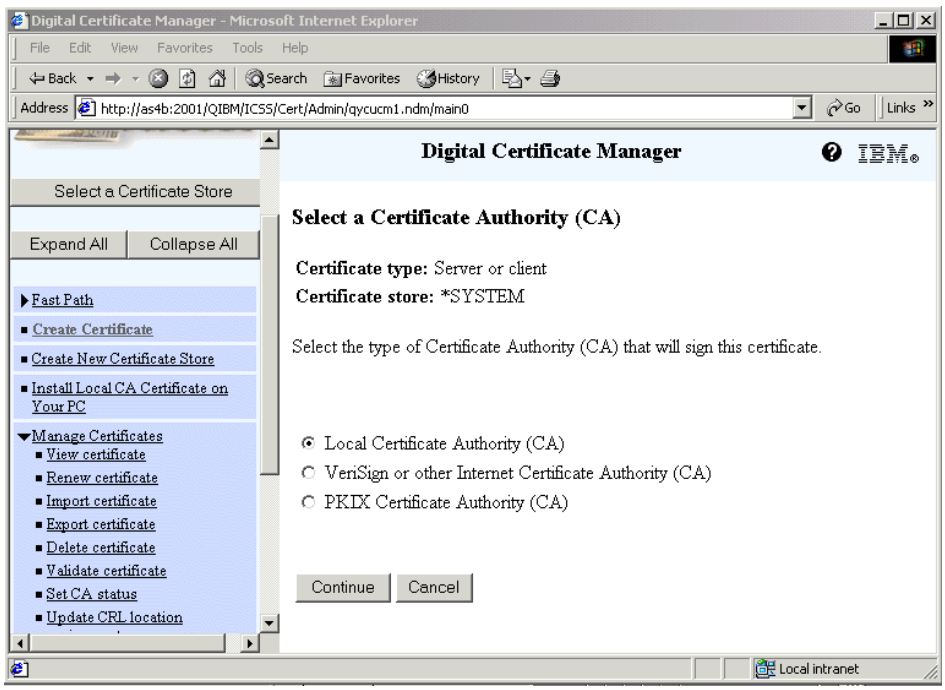


Figure 12. Select a CA in the Create a Certificate task

Before you go through the necessary steps to create the certificates, keep in mind that the selection of the CA that will issue your certificate depends on the application for which you will use the certificate. For example, for intranet applications, you may want to use a local CA to issue the certificates. But when you offer Web services to the public, you should use a well-known CA whose CA certificate is already shipped with most application clients. For more information, refer to *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659.

2.2.1 Requesting a certificate from a local CA

Generally, it's useful to use your local CA to sign certificates when you want to issue certificates to systems and users within a more limited scope, such as within your company or organization. Creating and maintaining your own CA allows you to issue certificates only to those users who are trusted members of your group. This provides better security because you can control more stringently who has certificates, and therefore who has access to your resources. A potential disadvantage of maintaining your own CA is the amount of time and resources that you must invest. When you use a private CA, each client application that uses SSL to communicate with a server must

obtain and store a local copy of the private CA's public certificate. Depending on the number of clients in your network, this may result in a higher management cost than purchasing certificates from a public CA whose CA certificate is often built into the client's certificate database.

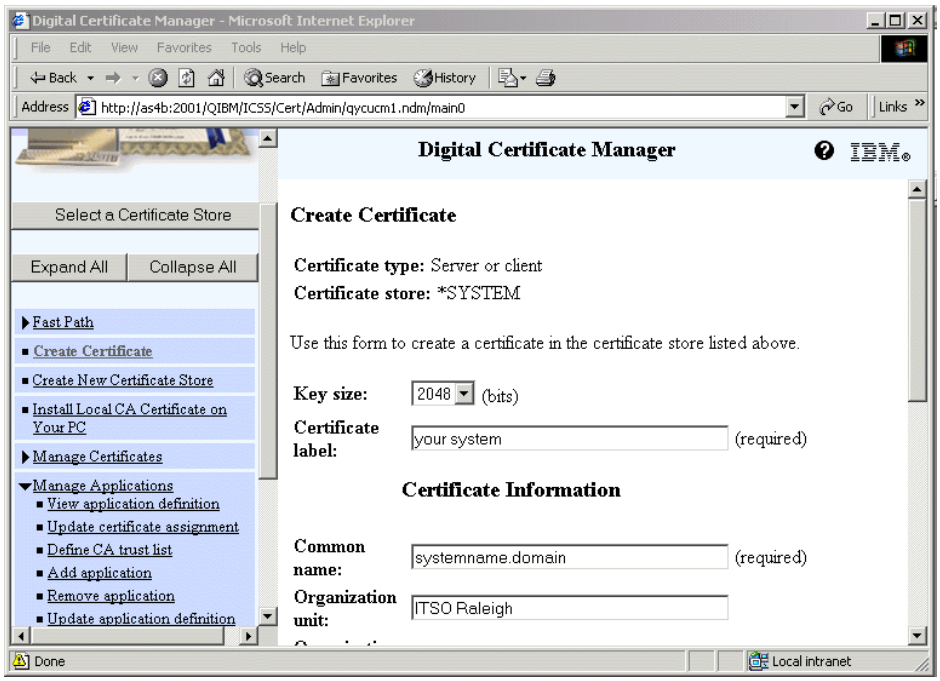


Figure 13. Create certificate locally signed (Part 1)

We assume that the local CA has already been created on AS/400 system.

1. In the window shown in Figure 12, select **Local Certificate Authority (CA)**, and click **Continue**. You will see the pages shown in Figure 13 and Figure 14 on page 37.

   Fill in all required parameters:

   - **Key size**: Select a key size to use for generating the certificate's public and private keys. For all certificates except user certificates, the selection box displays the key sizes supported by the Cryptographic Access Provider licensed program (5722-AC2, or 5722-AC3) on your iSeries or AS/400 server. For user certificates, the available choices also depend on the browser support. With the strongest cryptographic support, your choices are 2048, 1024, 768 or 512 bits. Refer to Chapter 7, “Ciphers and cryptographic product considerations” on page 373, for detailed information about the supported key lengths.

The larger the key, the more secure the encryption it provides. However, using a larger key size may degrade performance for a secure session.

- **Certificate label**: Enter a name for the new certificate. Because this name uniquely identifies the certificate in the certificate store, you must specify a unique label. This is a required field.
- **Common name**: Enter a name to describe the certificate application. This is a required field. How you choose a name for the certificate varies depending on the environment in which the application communicates data. For example, if you are creating a certificate for a server application that communicates data over the Internet, you should use the fully qualified domain name for the server. This ensures that browser software can validate the certificate correctly. When the certificate's common name does not match the URL domain name, the browser can validate the certificate. However, the browser cannot verify the server's identity. Consequently, the browser may warn the user of this discrepancy and prompt the user to decide whether to trust the certificate.
- **Organization unit**: Enter the name or numeric designation of the organizational section for the application that will use this certificate. This is an optional field.
- **Organization name**: Enter the name of the company or divisional section for the application that will use this certificate. This is a required field.

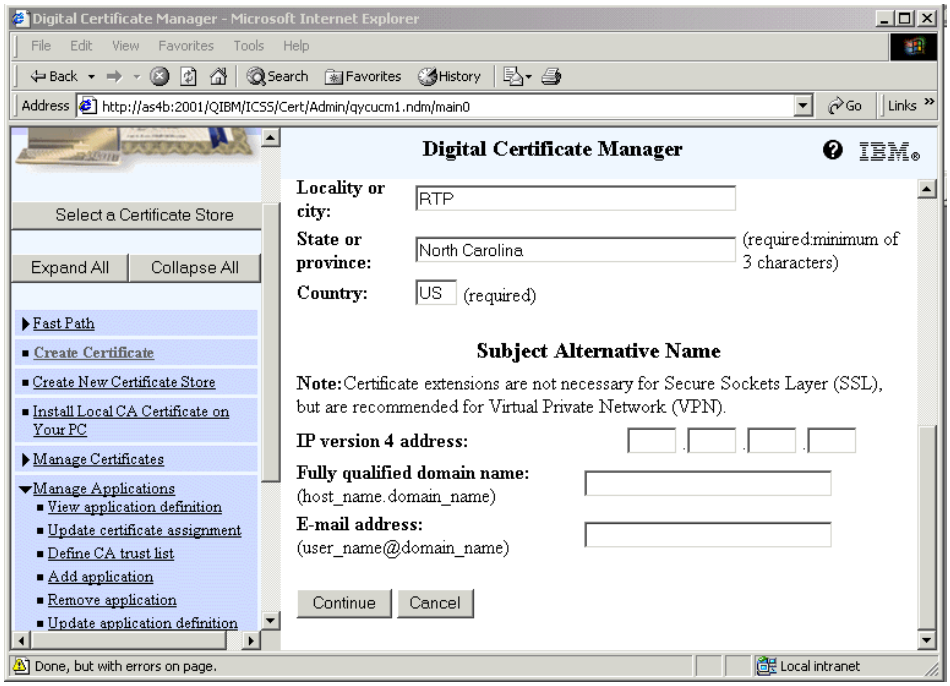


Figure 14. Create certificate locally signed (Part 2)

- **Locality or city**: Enter the name of your city or a locality designation for your organization. This is an optional field.
- **State or province**: Enter a designation for the state or province in which you will use this certificate. The state or province designation must be a minimum of three characters in length. This is a required field.
- **Country**: Enter a two-letter designation symbol for the country in which you will use this certificate, for example, US for United States, DE for Germany, or IT for Italy. This is a required field.

In V5R1, this page provides alternative name fields. These fields allow you to extend the identification information for a certificate. Providing information for these fields is optional if you are using this certificate for Secure Sockets Layer (SSL) communications. However, you should provide information for these fields if you are using this certificate to identify an AS/400 virtual private network (VPN) connection. These certificate fields correspond to the types of identifiers that you can select when you define an AS/400 VPN connection. To work correctly, the

certificate that you use for a VPN connection must have appropriate information in the field that matches the VPN identifier.

- **IP version 4 address**: Enter your IP address. If you are using the certificate for a VPN, you should enter the IP address of the connection endpoint that this certificate identifies.
- **Fully qualified domain name**: Enter the fully qualified domain name for the application that will use this certificate, in the form `'hostname.domainname'`. If you are using the certificate for a VPN, you should enter the domain name of the connection endpoint that this certificate identifies.
- **E-mail address**: Enter your e-mail address, for example `'sjones@ibm.com'`. If you are using the certificate for a VPN, you should enter the e-mail address associated with the client connection endpoint that this certificate identifies.

2. Click **Continue** at the bottom of the page.

   DCM uses the input fields to create the server or client certificate, this task may take a few seconds, depending on the AS/400 system model. You will receive a completion message and the list of applications as shown in Figure 15.

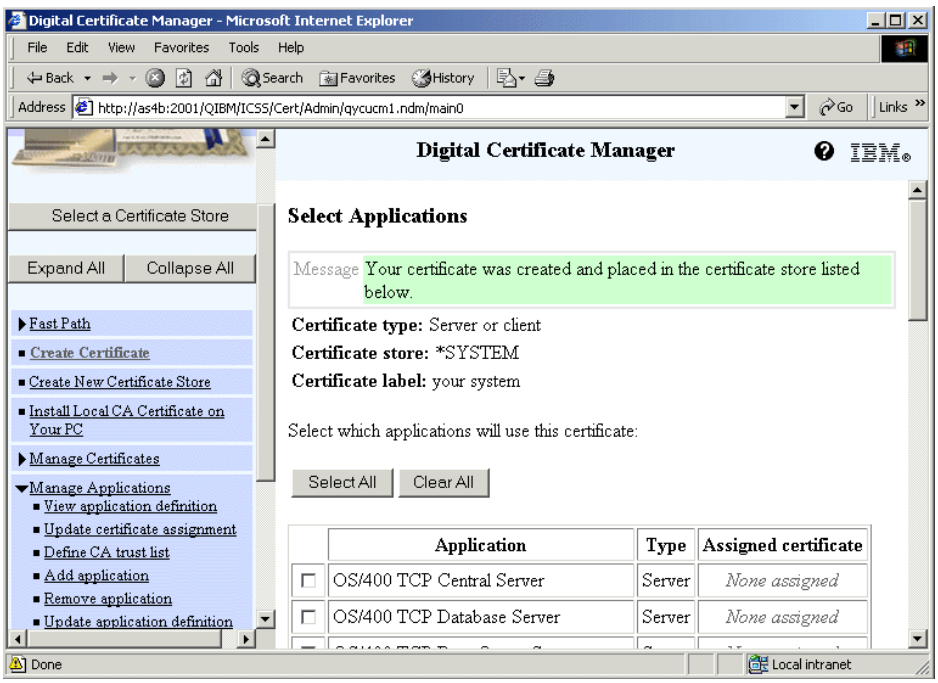


Figure 15. Select Applications task

3. Select the applications that will use the certificate for SSL session, and click **Continue**.

   The table lists each available application, indicates the type of application, and shows the current certificate assignment for it. You can assign the certificate to more than one application by selecting multiple applications from the list. You can skip this step now or review your choice in the Manage Applications task.

   Remember to distribute the CA certificate to all clients that are communicating with applications that have this server certificate assigned to it. A more common way is to allow users to access a Web page from which they can receive the CA certificate into their browsers. Users that have an AS/400 user profile can connect to DCM and receive the CA certificate through the Install CA certificate on your PC task.

   You will receive another completion message.

4. Click **OK** to finish the task.

Now you are ready to use the server or client certificate issued by the local CA to secure applications. For the TCP/IP servers, before you can actually

establish an SSL connection to such a server, you must configure the server for SSL, and if necessary end the server and restart it. After the server is restarted, it listens on a different port for SSL connection requests.

For an example of how to secure your applications, refer to *Securing applications with SSL* found in the iSeries Information Center by clicking **Security->Securing applications with SSL**. Or see *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659.

2.2.2 Requesting a certificate from a well-known CA

You may prefer to use a public well-known CA to issue your certificates when you allow general public access to an application, and users may already be using certificates from a well-known CA for other purposes. Therefore, choosing certificates from a well-known CA creates less of an administrative burden on both the affiliated organization and your system administrator. Public Internet CAs issue certificates to anyone who pays the necessary fee. However, an Internet CA still requires some proof of identity before it issues a certificate. This level of proof varies, however, depending on the identification policy of the CA. You should evaluate whether the stringency of the identification policy of the CA suits your security needs before deciding to obtain certificates from the CA or to trust the certificates that it issues. The terms and conditions under which a CA issues certificates are described in the Certification Practice Statement (CPS).

Figure 16 shows a flow chart for obtaining a server or client certificate from a well-known CA.

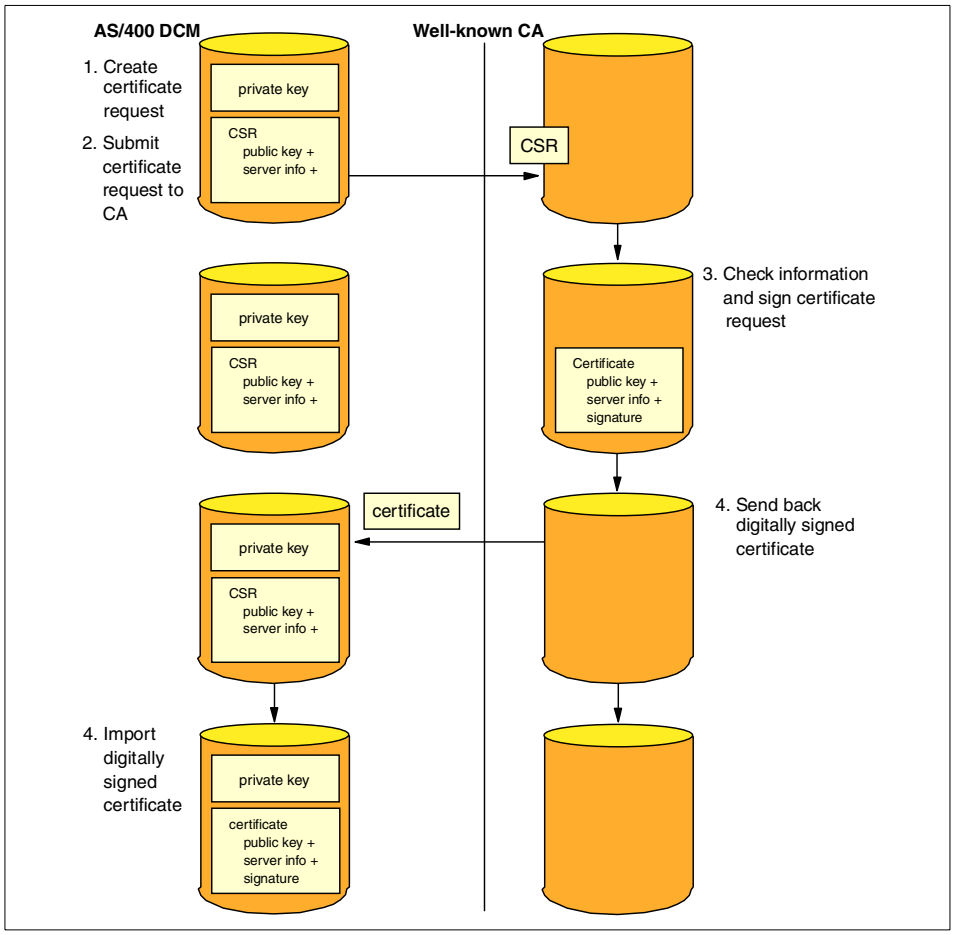


Figure 16. Obtaining a server or client certificate from a well-known CA

This operation creates a private key that should never leave the owner of the request. It also creates a certificate request, often referred to as a Certificate Signing Request (CSR), which is a digital certificate that has not yet been signed. It contains the public key and the distinguished name. Then, the certificate request is submitted to a public CA, which digitally signs the certificates and sends it back to the requester. The server or client certificate can be imported into the certificate store to use.

Now let's see all the steps in detail. Referring to the window shown in Figure 12 on page 34, you must select **VeriSign or other Internet Certificate Authority (CA)** and then click **Continue**. Then you see the window shown in Figure 17 on page 42.

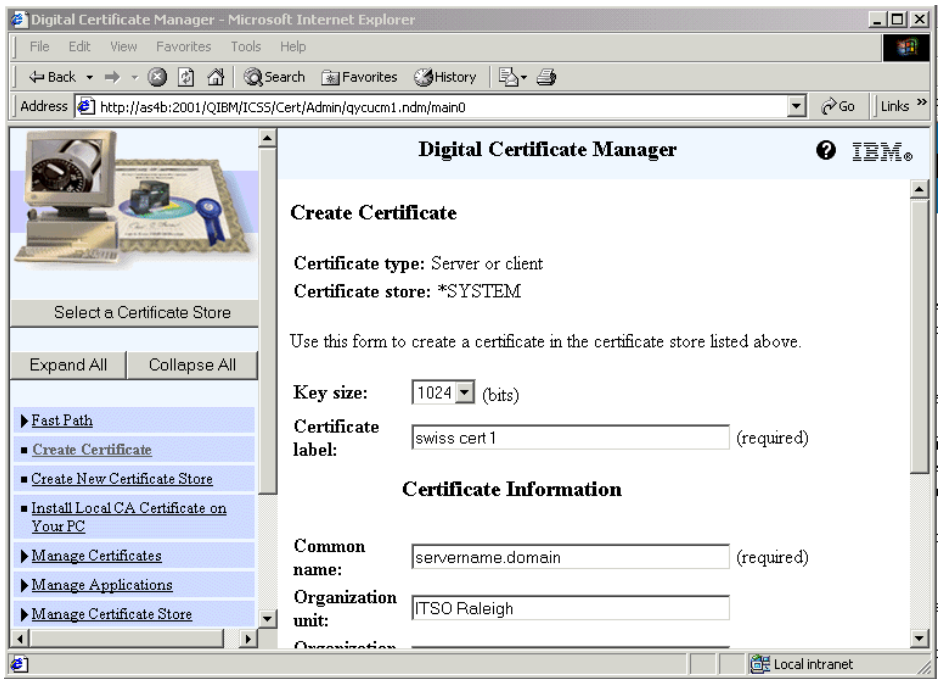


Figure 17. Creating a request certificate

1. Fill in the form.

   Refer to 2.2.1, “Requesting a certificate from a local CA” on page 34, for details about the required parameter. Note that when requesting a certificate from a well-known CA the Subject Alternative Name fields are not displayed in the form shown in Figure 17. A public CA expects to receive a PKCS#10 certificate request in Base64 encoded format. The PKCS#10 certificate request contains a standard X.509 certificate template without any extensions. If you need this extended identification information for a VPN implementation, check with your CA provider about whether you can specify them during the module completion after pasting the PKCS#10 request into a buffer on their Web site. If the certificate is created by the public CA with these additional extensions, once you import it into your certificate store the extensions will be available for viewing.

2. Click **Continue**.

   DCM creates the key and the certification request. The request data consists of the public key and other information that you specified for the new certificate. This information is sometimes referred to as a Certificate Signing Request (CSR).

You receive a confirmation page, as in the example in Figure 18, which displays the request data that you must provide to the public Certificate Authority (CA) that will issue your certificate. DCM places information about this pending certificate request in the current certificate store. You can display it using the Management Certificate task.

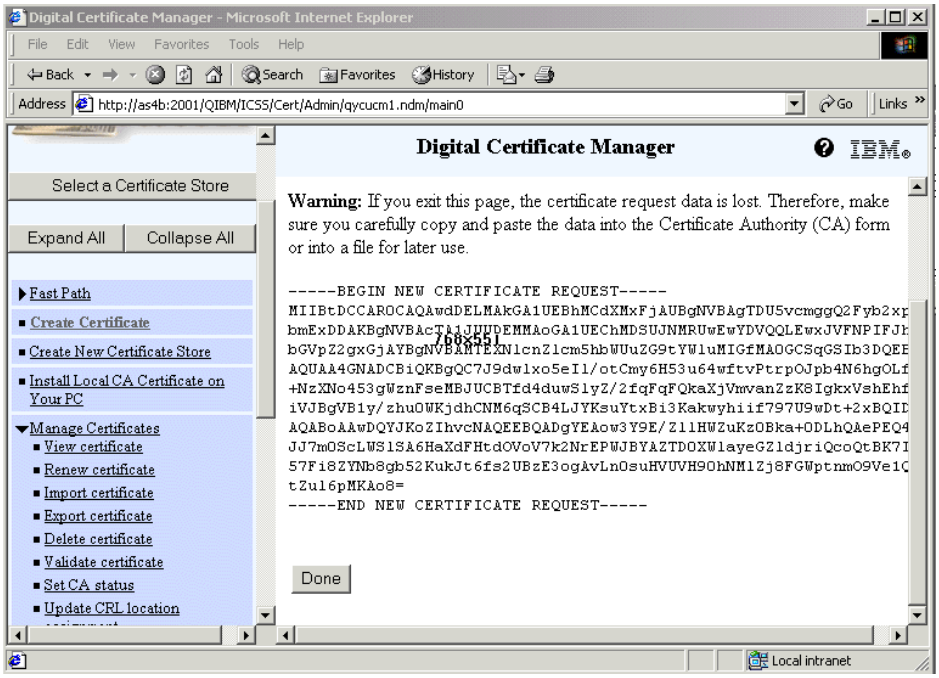


Figure 18. Certificate request created

3. Carefully copy and paste the request data into the form, or a separate file, that the public CA requires for requesting a certificate.

   You must use all the request data, which includes "`-----BEGIN NEW CERTIFICATE REQUEST-----`" through the area that ends with "`-----END NEW CERTIFICATE REQUEST-----`". The certificate request data is prepared for requesting the certificate from the CA.

   Since some CAs charge a considerable amount of money for the certificates that they issue (or that they re-issue), we suggest you back up your certificate store. Refer to 2.7, "Backup considerations" on page 93.

4. Submit the certificate request to a well-known CA.

   The way to submit a request depends on the CA. You may have to send the request data via e-mail to the CA or submit the request online over the Internet.

In this case, the certificate request is submitted online over the Internet choosing VeriSign as the well-known CA. For demonstration purposes, a trial certificate is requested.

To request a certificate from VeriSign, enter the URL:

`www.verisign.com`

Then, follow the instructions given on this Web site. After you provided the necessary information requested by the CA, both for the system and for the administrator, you see a form to paste in the certificate request data as shown in Figure 19.

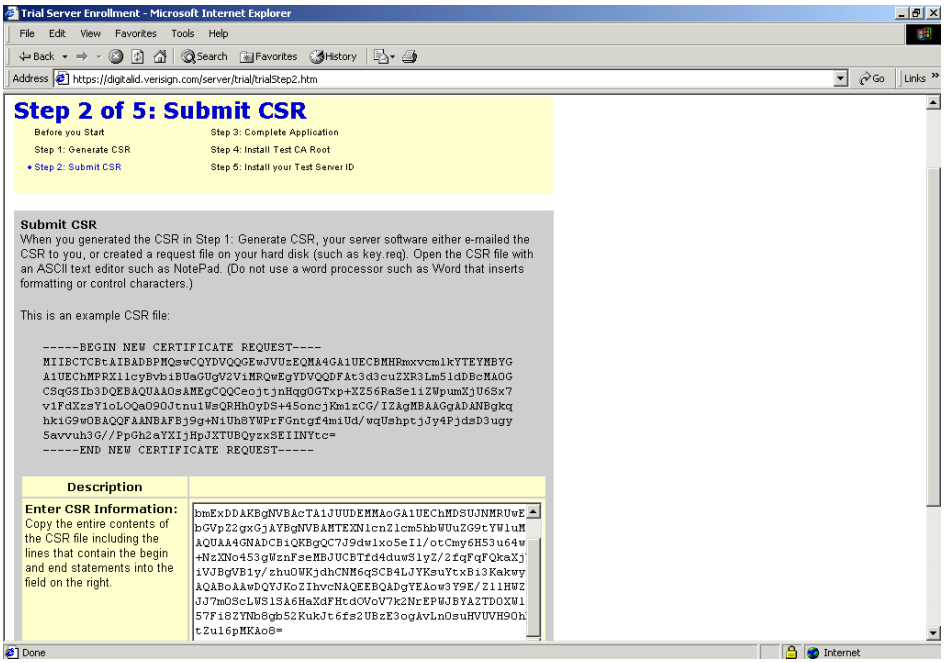


Figure 19. Submit CSR

After some time, the administrator who submitted the request receives the certificate through e-mail as shown in Figure 20. Note that there are different methods in place for obtaining the certificate data. For example, some CAs only send e-mail containing a certificate identifier. Then, you have to go to a specific URL to obtain the certificate data from there.

Depending on which CA you are dealing with and on the class of the certificate requested, the checking and time to issue a certificate are different. You have to refer to the CA's Certificate Practice Statement (CPS), which contains the terms and conditions under which a CA

operates. Since this request is for a trial server certificate only, no further checking and approval work is performed by the issuing CA.

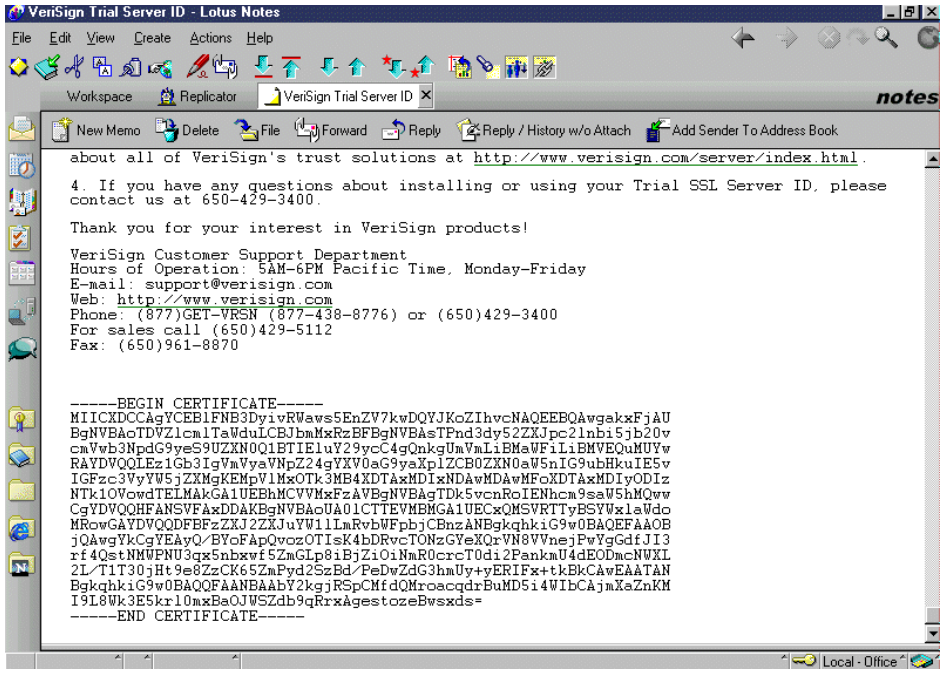


Figure 20. Certificate signed by VeriSign

5. Copy and paste the certificate data into a text file on your PC using, for example, the Notepad editor or another ASCII text editor.
6. Transfer the PC file to an IFS directory on your AS/400.

   You can use FTP or map a directory to your AS/400 to store the file in IFS or use the Operations Navigator. However, make sure that you do not use binary mode when transferring the file to the AS/400 server.
7. In the DCM task, click **Select a Certificate Store**. Choose your certificate store contains the pending request and click **Continue**.

   In this case, it is *SYSTEM as shown in Figure 21 on page 46. In the next window, DCM asks you to insert the password of the certificate store, and click **Continue**.

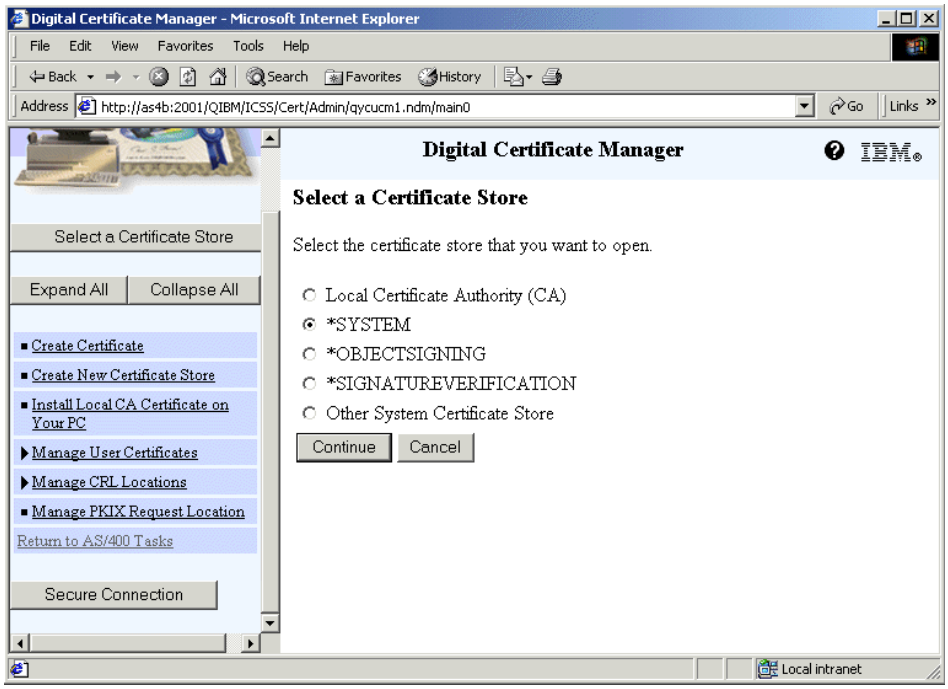


Figure 21. Select a Certificate Store task

8. When prompted, enter the certificate store password, and click **Continue**.
9. Click **Manage Certificates** to expand the task, and then click **Import certificate**. Select the type of certificate you need to import.

   The window shown in Figure 22 appears. Remember that you can receive your server or client certificate only if the issuer of your certificate is in the list of trusted CAs. Otherwise, you have to import the CA certificate first and then proceed with the server or client one. This could happen when the issuer of the certificate is an intermediate CA (refer to 2.1.4.1, “Certificate Authority (CA) certificates” on page 30).

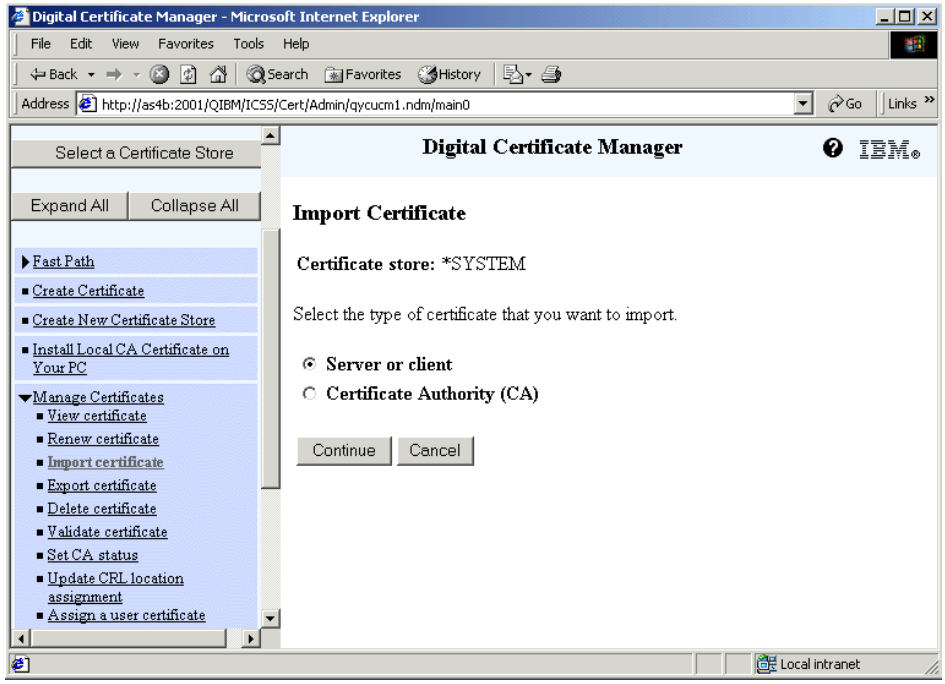


Figure 22. Import Certificate task

The CA usually provides the link where you can find the CA certificate, when it is necessary, in the e-mail with your signed certificate.

For example, in our case, the e-mail received indicated that before using the Trial SSL Server ID, you must install the Test CA Root in each browser that you plan to use as part of your test of SSL. To download the Test CA Root, go to: `http://digitalid.verisign.com/server/trial/trialStep4.htm`

Then, follow the instructions on that page to install the Test CA Root on the browser. Complete the following steps to install the CA certificate on DCM:

a. Obtain the CA certificate from your CA

   From your browser, enter the given URL and follow the instructions. You need to save the CA certificate to your local disk.

b. Transfer the PC file to an IFS directory on your AS/400.

   You can use FTP or map a directory to your AS/400 to store the file in IFS or use Operations Navigator.

c. Open the *SYSTEM certificate store in DCM, and expand **Manage certificates**.

d. Click **Import certificate**.

e. Select **Certificate Authority (CA)**, and click **Continue**.

f. Insert the fully qualified name of the ASCII file containing the CA certificate, and click **Continue**.

g. Insert the CA certificate label, and click **Continue**.

   Enter a name for the CA certificate that you are importing. That will be the unique identifier of the CA certificate in the certificate store, so you must specify a unique label.

h. Click **OK** in the completion window that appears.

   You can view this certificate in the Management Certificate task. Remember, steps a through h are required only when the CA certificate that signed the certificate request is not in the certificate store.

Now you can continue importing your server or client certificate.

10. Select the **Server or client** option, and click **Continue** in the window shown in Figure 22.

11. Specify the fully qualified name of the file that contains the server or client certificate that you want to import as shown in Figure 23.

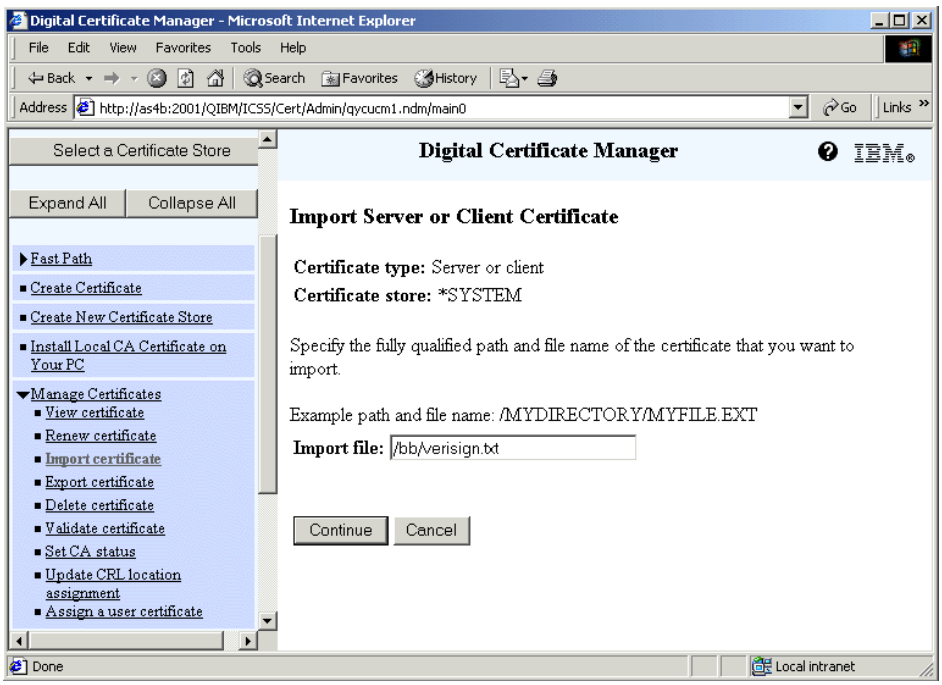


Figure 23. Import server or client certificate

12. Click **Continue**.

    You will see a window with a confirmation message.

13. Click **OK** to return on the task list.

Now you can verify your certificate in Manage Certificates task as a server or client certificate. You can also assign it to your application in the Manage Applications task.

Now you are ready to use the server or client certificate issued by a well-known CA to secure applications. For the TCP/IP servers, before you actually can establish an SSL connection to such a server, you must configure the server for SSL, and if necessary end the server application and restart it. After the server is restarted, it listens on a different port for SSL connection requests.

For an example of securing your applications, refer to *Securing applications with SSL* found in the iSeries Information Center by clicking **Security->Securing applications with SSL**. You can also refer to *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659.

2.2.3 Requesting a certificate from a PKIX location

A Public Key Infrastructure X.509 (PKIX) CA issues certificates based on the Internet X.509 standards for implementing a Public Key Infrastructure. PKIX standards are outlined in Request For Comments (RFC) 2560.

To understand the difference between requesting a certificate to a well-known CA compared to a PKIX CA, keep in mind the scheme shown in Figure 16 on page 41 and compare it with the one shown in Figure 24.

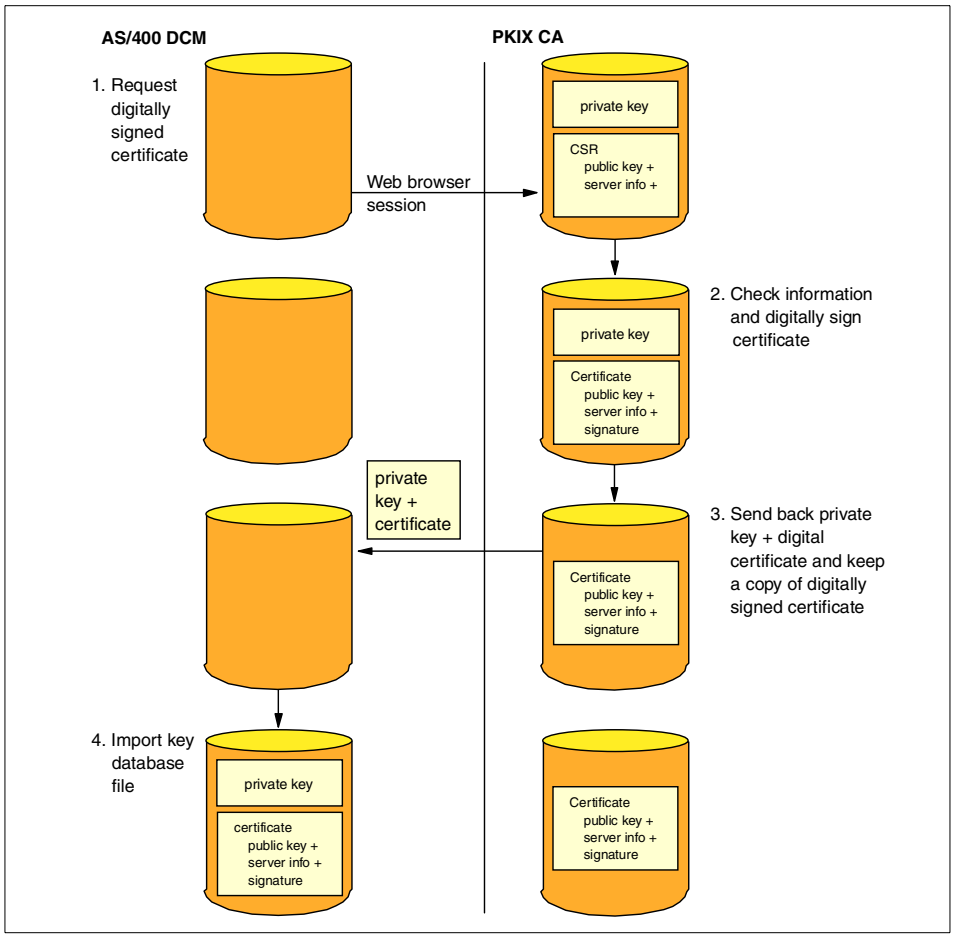


Figure 24. How to obtain a server or client certificate from a PKIX CA

When you want to get a certificate from a PKIX CA, you do it directly on the PKIX CA location. The key database file, containing the private and public key and the information related to the requester, is on the PKIX machine. Nothing

is done or created on the AS/400 system at this time. The PKIX CA checks the information provided by the requester and eventually signs the certificate request, as the well-known CAs do, based on its Certification Practice Statement (CPS). Afterwards, the PKIX delivers the digital certificate and the private key to the requester keeping a copy of the requester's digital certificate on its machine. The administrator of the requester system has to import the key database file into a certificate store to use it.

The fact that the PKIX CA holds a copy of the digital certificate in a directory is the essential difference between the other CAs, because the public key is now accessible to everyone who has access to the PKIX system. To encrypt a message for user A or to verify a digital signature from user A, user A's public key is required. For this to work in an organization, everyone must have access to all the public keys.

The PKIX server typically provides this facility. For example, Lotus Domino provides a PKIX type CA server for public use. The secure mail is a typical implementation of how it could be useful to have a server that shares the certificate of the entire organization. When user A has to send secure mail to user B, but it doesn't have the public key of the recipient, user A can query the PKIX server to obtain the digital certificate required. Certificates are usually published in an LDAP directory. A PKIX CA could be either public or private. The criteria to choose one instead of the other is the same for the public and private non-PKIX CA.

Generally, a public PKIX CA requires more stringent identification before issuing a certificate. Usually it requires an applicant provide proof of identity through a Registration Authority (RA). After the applicant supplies the proof of identity that the RA requires, the RA certifies the applicant's identity. Either the RA or the applicant, depending on the CA's established procedure submits the certified application to the associated CA. As these standards are adopted more widely, PKIX compliant CA will become more widely available.

If you choose to have a PKIX CA issue certificates for your applications to use, you can use Digital Certificate Manager (DCM) to manage these certificates.

You can use DCM to configure a URL for a PKIX CA by selecting the **Manage PKIX Request Location** task. Doing so configures Digital Certificate Manager (DCM) to provide a PKIX CA as an option for obtaining signed certificates, as shown in Figure 12 on page 34. Starting from that window, to obtain a digital certificate, you must select **PKIX Certificate Authority (CA)**, and click **Continue**. Then, you see the window shown in Figure 25 on page 52.

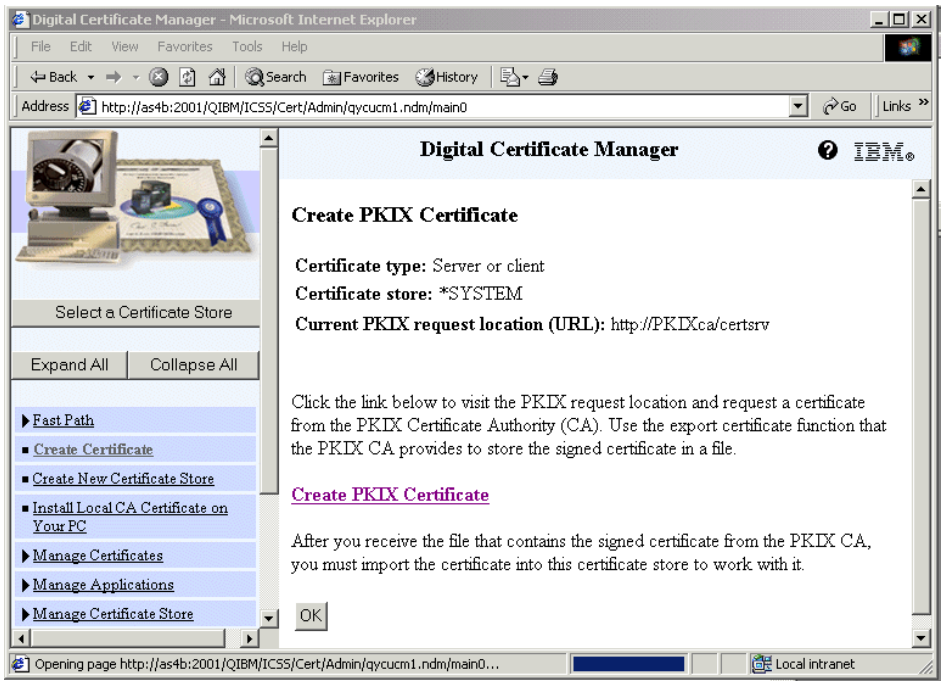


Figure 25. Create PKIX certificate task

The current PKIX location is the URL specified in the Manage PKIX Request Location. Note that there is no form to fill in, only a URL linking the PKIX CA. If you click **Create PKIX Certificate**, it opens another window browser with the PKIX URL, as shown in Figure 26. On the AS/400 system, now you can click **OK** and close your DCM working session, because the creation task of the certificate runs on the PKIX CA Web site.

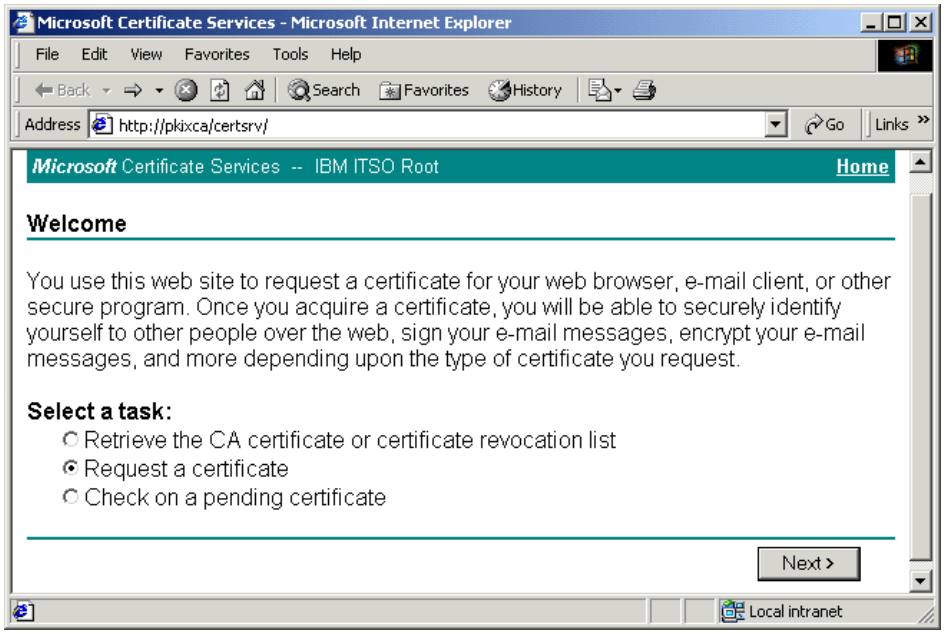


Figure 26. PKIX CA Web site

After completing all the steps required by the PKIX CA for obtaining a certificate, the PKIX CA creates the public key pair, creates the certificate, and signs it. When you receive the signed certificate including the associated private key, you must import it into a certificate store. Manage it as a digital certificate created and exported from another system. Before importing the keys, check whether the CA is known by the DCM. There is no attribute on the certificate to make DCM aware that it was created by a PKIX CA instead of a non-PKIX one. The additional value provided by the PKIX CA is that it publishes the digital certificate and thus makes it available for others to use. For information related to the import function, refer to 2.3.1.1, “Importing certificates” on page 57.

2.3 Managing certificates

Use the Manage Certificates page to choose the type of management task that you want to perform for certificates in the current certificate store. This task is not available in the local Certificate Authority (CA) store.

The types of tasks that you can perform depend on the certificate store with which you are working. This section briefly covers the tasks that are available

when working with the *SYSTEM certificate store. The focus is on the enhancements introduced with OS/400 V5R1.

- **View certificate**: This task allows you to view a distinguished name and other information for a certificate in the current store. You have to select the kind of certificate you want to see. Notice that in V5R1, you can also display the pending certificate requests you sent to a well-known CA. If the issuer of the certificate includes the extension fields, such as the subject alternative name and the key usage, you can also see them.
- **Renew certificate**: This task allows you to renew an existing, expiring certificate. When you choose to renew a selected certificate, the DCM asks you who signed the certificate. In V5R1, you can choose how to handle the renewal of the certificate that you obtained from a public Internet Certificate Authority (CA). You can select to create a new public-private key pair for the certificate that you are renewing or to import the renewed certificate from an existing file. This option assumes that you have already received your renewed signed certificate from the issuing public Internet CA.
- **Import certificate**: This task allows you to add a certificate to the current certificate store. Refer to 2.3.1, “Importing and exporting certificates” on page 55.
- **Export certificate**: This task allows you to copy a certificate from the current certificate store to a file or to another certificate store. Refer to 2.3.1, “Importing and exporting certificates” on page 55.
- **Delete certificate**: This task allows you to delete a certificate from the current certificate store or to remove a certificate from a specific user profile. Similar to the view task, starting with V5R1, you can delete a pending certificate request from a public Internet CA in the current certificate store.
- **Validate certificate**: In V5R1, you can validate the authenticity of a certificate (server, client, and CA) in the current certificate store. By using the default when validating a certificate, a CRL is checked when a CRL location has been assigned to the CA certificate that issued the certificate. If there is no CRL location assigned, CRL checking is not performed, but the certificate is still subject to the normal validation process. During the validation process, DCM checks that the selected certificate is not expired. DCM also checks that the CA certificate for the issuing CA is in the current certificate store and that the CA certificate is enabled and marked as trusted. If the certificate has a private key, DCM also validates the public-private key pair to ensure that the public-private key pair match. In other words, DCM encrypts data with the public key and then ensures that the data can be decrypted with the private key.

- **Set CA status**: This task allows you to enable or disable a CA certificate in the current certificate store. If you enable a CA certificate, you can use certificates issued by that CA to establish secure sessions. If you disable a CA certificate, you cannot use certificates that the CA issues to establish secure sessions.
- **Update CRL location assignment**: In V5R1, you can add or change the CRL location assignment for a CA in the current store. Refer to 2.5, “Exploiting the Certificate Revocation List support” on page 82.
- **Assign a user certificate**: This task allows you to assign a user certificate that you own to the user profile under which you are currently working (signed on) in DCM.
- **Update device assignment**: In V5R1, if you have a 4758 PCI Cryptographic Coprocessor for iSeries installed, you can change the cryptographic coprocessor device assignment for a certificate that uses a coprocessor either to store or to encrypt its private key. Refer to Chapter 4, “Using hardware cryptography support for SSL/TLS” on page 189.

2.3.1 Importing and exporting certificates

Transferring the certificates between the AS/400 and other systems, which could belong to different platforms, is an essential function in managing the digital certificates. You may want to import a CA certificate for client authentication purposes, or you may want to create and export a server certificate with DCM for another system.

The examples could be as many as the possible needs and environments are. For this reason, it is fundamental to understand the different standard supported by the AS/400 system and when they are used. Because certificates can be encoded differently, be aware when transporting the different encoded certificates to and from the AS/400 system. There are two main formats used by the DCM to handle the certificates:

- **The Base64 encoded format**: Base64 is an ASCII format to encode the digital certificate. The encoded data is placed between the `----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` lines as you can see in Figure 27. DCM uses this format to import or export the certificates without their private keys, as when you copy your local CA certificate on a PC or when you import a signed certificate requested from a well-known CA. Base64 is a text format and must be transported as text applying the proper translation. Therefore, you can copy and paste the certificate data into a text file, or you can use the FTP with ASCII type format to convert it.

```
-----BEGIN CERTIFICATE-----
MIICXTCCAcagAwIBAgIEOoarZjANBgkqhkiG9w0BAQQFADBzMQswCQYDVQQGEwJV
UzEXMBUGA1UECBMOTm9ydGggQ2Fyb2xpbmExDDAKBgNVBAcTA1JUUDEMMAoGA1UE
ChMDSUJNMRUwEwYDVQQLEwxJVFNPIFJhbGVpZ2gxGDAWBgNVBAMTDO1UUO8gUmFs
ZWlnaCBDQTAeFw0wMTAyMTAxNTEwMzBaFw0yMTAyMDYxNTEwMzBaMHMxCzAJBgNV
BAYTAlVTMRcwFQYDVQQIEw5Ob3J0aCBDYXJvbGluYTEMMAoGA1UEBxMDU1RQMQww
CgYDVQQKEwNJQkOxFTATBgNVBAsTDE1UUO8gUmFsZWlnaDEYMBYGA1UEAxMPSVRT
TyBSYWxlaWdoIENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDsDJe1J2Ef
5rKi7DGPlqXdHHqrP92i8FFNDMkmJBnbU/OT3ljrb+7/ZqoyaEfFiIiXcWCODUVS
SKZzy2ocF1M/F1Fr1XfdrcKK5HddiG8wpiucCIJDL/gveJOopgiKAtKWExvWRu7t
45u4nptUDzqxoGhSMx2Difze/DAjtYkCnwIDAQABMA0GCSqGSIb3DQEBBAUAA4GB
AITBY3FxDB1KV7jhR/anN69UX4y7AGMCnPI2Y7O+2mzlW25vb+Evv5PoQZTTLv95
wsUWEaSVpLV5FHyvV9H5OyE+4gFfxSRP55teVJq4oIDcsLVUgPAc2t1ddWPogkco
+9EavbnEwsKoEOPeLVTM8dV6sfpis9PKpd+fCfBGvYSb
-----END CERTIFICATE-----
```


Figure 27. A Base64 encoded certificate

- **The PKCS#12 DER format**: PKCS#12 is also known as Private Information Exchange (PFX). It is a binary format to encode the digital certificates as shown in Figure 28. One or more X.509 certificates and private keys could be contained in a PKCS#12 DER encoded certificate file. DCM uses this format with the importing and exporting function of the client or server certificates with their private key.

```
0,□é□□□□0,□@□    *†H†÷□□□□ ,□š□,□-0,□'0,□Ž□      *†H†÷□□□□ ,□□0,□{□□ 0,□t□
*†H†÷□□□□0□□□*†H†÷□□□□□0□□
□<lxsµ□"÷□□□€,□H¦ù|□†¢□/ÍBxR(ÀÇð"»□H□«Ç…¤ë□K6□Å□ó M□„Y□„ò œ`Œ"82m□¨~ž3Ù°äzU□Y<ª„¤Ÿ¶Û÷
NÆ˜xU"¸š´¼Ã     tä¡8<\¯□&â;„~J<F Òï!M.Ð'hfJ□%ÕÑñb÷□—æ•Ž%"#®□{ô  "ä¶e<□□í
¤2¼~b›Żu$ù)□E¨^É_□ Q□□N…ZÝ<:pA"P¦¾þN/¨Ï°'<n IÊÑÕ□ÉÉq□□VÖß«□□
□¯-¿Ê¥E®Ïf¡□"»q@£-„[uP□¶ÿ°□yØIa—‡T°(S¨'¿oA•]ä9ð¿^Z
Û%GÀ¤LÒª™YUÈ□1„□%è°£ãÍð»m}ÓÍnNâ$□âü«ÿí§□□□OÙÝ□ÀÙÝÕ□¦rÈ»¸çsb€□ËÒ_□ø]ò
r••ÔŠ"□□D„Å□}ŽÊAúÊM:ÿ{}□□3rªÝÀ□•ÙWcG&ë~¶±XMe)@ª□Š'ª Ó¿J□ÌCGM¾›n¡±)aÈU0□,
u´ª^□t□ü®it'"¿j¿wš@wÝßBÅ‡□È<2¬¦¢7v□ÝafÐñ¼^!Ùþ Ô¼J□□□†¿í ª Ž}G†Ý@@ □S"□
HÅá+□□□ÅÝüðt=®N•!CHFÎÉÍÍªÇ□fêðTG¦□Ó>□Éé¥˜ª.…f□(é-¦Òà-ÓP'□□ tÍ„Ó›¡A°91Dc`Àœ□
□¤□□5Æ-ÛY½ÿ□□□ÈUQÖ!?9HOï£'Z'â□□|Ðê*9•¼¯Þ/î□H«3¡□;P/□QÔ
qWOg'ªD$/®\ŠJ¼®g□Ú¤>YÚ•&ï"Ÿ—ZS/î ªªÅdsç!„Ùï Ó□ÜmyI□ÎY;□8˜ □ÔÝn)÷DC□‡'ÒÈJÛâØ□LÓ¬GpI□£
-%×¡3ÇÅ□4Ãì "½½ìkZ¤EÛ□B%YØV\`x5Í6´□?âãñ—ó_±ÉÓÊÚÝŽ□-Ý"y□)&O□□»
£□□É}□$□□□<Å¼ª3Ø>ð:□□Í□áß×|*"Ê†~§mJ□□h%üÐGn‡Ô|óÑ□Í ªÙS°îzð`W□ª„
wÏ3ÚÅq®˜}¾óóÍ¢P□wD?g"ÌÒ%ñ¶Å□}í•^B÷Ö□rÙ¬ª
ZÙó-â¼W□Šë□íÅÉ¥Š□¥□8□NÞ±†Æ□ècgž¦|ëh=¿,'Iwª)ª'□?(Ô>‡¦H%¬~‡□e%g—r®□¼5[ÏŠŸÞ□□
¶¥ø□À□—Š□Š-ýªdÉ□□ª+^□
°h8G7âÑçŒîÝXÈ'˜2c8Å□~«)éÐ| □9y€<_!>□^□ìvw„ŸÖI]□ÅY„□Å□□Ú'Ó6z\□FKÓL□8|□Š8Û4ä`jb□"ÑF¤ì
Of□{KyêÈ□ßêËÅ—  ÐÌ¸Ð¡Í>ª$´:ÉÊs□âÊ÷ooufì□!à□9Œ]æ±□<[,□□
"V›Z9þ¶¯)Œ~ëo¨"Ï¼ÜŠ@æ¼•□Ù□□□-'ï@"I□PÓ¼'¸□•Å8>Ï"G±…QV[£ò"±ììAfÙÏø"| .êãÓ¥4QY°wòÅJöÓ□
Åµ4ÚØP'°tæ□u□^7¤÷Þ¬ªœîÊ□H¸-äÜä□Î¿Ük¡Ö:hÛ˜□|Œð¿n,úäüz8c÷äv'EÀ□íeŽÄ□5 uã`J È»¿@Ê=□
5÷©□".T□T□¦.%r4°ÐÉ:     4Ù…□°É□˜9ß<MÎ□□•Ú®+ O□±;ÑÓ\Ù□□¨H□¶□y□÷Æ□I'□¿
~p□¤□8□ÍM_èÙ□cæ'bf9ª½;cCI□r□—'.*Þ□□"Ko¯¬ô¤□□+3ÃÙ—X¡'□£§?Mì¨~DSN„'H=□^¿8g=¬□îN…A□
Œ~ÅÂ¡□,¬_6ŒR}E`¯=D;□L…}=□—3ÂnÑwJî<RÊªLóvL@ã^Æ%0$ÜÓÓ¡□ñ™□E>{z(§□n□;^ÖM5µ'□•□□
□"ý"ÅÀŠ]T□"u□íJ.%"'OQ¤ï-□a¿-Ú‰ƒ[$<ZK´□ÅÎ□°□Ó□ó½"&:)')BGÅ§[…ÔßM›y□□¬□1¬□KDÎpo□
ò□5\`ìxgæ□ö'□Ó@□ºSbiè9¸Ç,□2èÇw¥®——~Ú[ÛýV□àñ»yMV>UñÊíÙ½áx•ê?ÓÍñí□□!£f;8é"f‡&¿Íí÷d□s-
\PŠÓ□@†"FefÊ^□□
SYH~/èyTÊøVÖ¥"âSƒ-ðßG□ìò^Z9w□züZòcÍ°b#™%k^Ù□ù-jÏw□l□q+\Öƒ□-(%□□ÝŽür°ÔÓ7'ýn8${™†Ã¥ýÈ
ÍÄ:NêÏ¶±+&Cæ{Ç  ÕÊ{VÌ¤Ó<£r…Rý8^†œ3%@Ã□Æs‡çì□U|LFW□□ä    gC¼E□êz'"□á
¼-Ó€ó˜:cn°zuK ªK"?Bzs•ÅêE¨^>YZ•µà)¡)„—□‡ŸHÉOÅYYÉ"É[ÖF¼P□ BèuMö×A□ì½G□úŒpÀ□□□x□øÞ¡þs¼
N□□\vLðø'"x •tO7O□□□□+□□□□□□Pl+ÛQÅ€h§Ì□ÅhxÖ[ƒEÇ`□□"˜□Ò¬[Å6K¿Åì0:□eƒäbÅ
```


Figure 28. A PCCS#12 encoded certificate

The PKCS#12 DER format is binary. Therefore, it cannot have an ASCII or EBCDIC translation. It must be transported in the exact binary format. You can use the drag-and-drop function of the PC, or map the IFS directory, or use the FTP with a BIN type format to transfer the image of the file.

The following sections cover the main steps used in importing and exporting certificates.

2.3.1.1 Importing certificates

You can import both the CA and server or client certificates as shown in Figure 29 on page 58. DCM automatically recognizes the format used for the encoded certificate you want to import and it asks for the password used to encrypt the data.

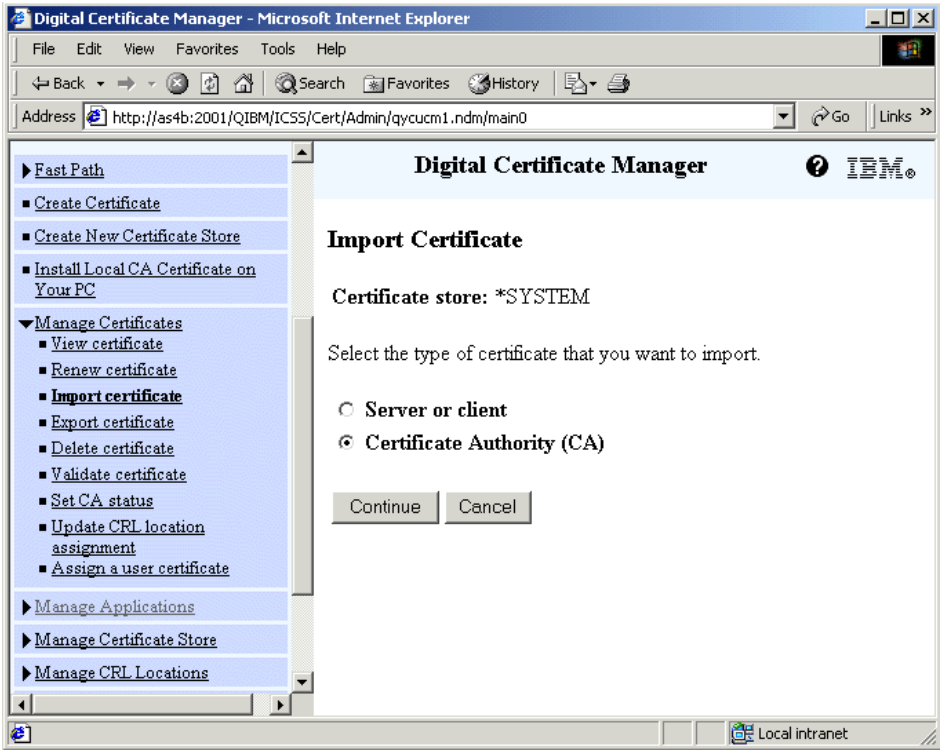


Figure 29. Import Certificate task

Let's discuss importing CA certificates. There are some situations when the CA certificate must already be present and trusted in the certificate store, such as:

- When you want to trust the client certificates issued by a particular CA during the SSL handshake
- Each time you want to import a signed certificate requested from a CA

When a hypothetical CA does not belong to the list of the well-known CAs present in the CA list of the DCM, as could be the case of a private CA or an intermediate one, you need to import it.

The Base64 format is usually used to transfer the CA certificates. How to obtain the text file containing the CA certificate in one of those formats depends on the CA itself. Once you have the file saved in an IFS directory, you can import it. Complete the following steps to install the CA certificate on DCM:

1. Obtain the CA certificate from your CA.

   The CA usually provides a link where you can find the CA certificate.

2. Transfer the PC file to an IFS directory on your AS/400.

   You can use FTP or map a directory to your AS/400 to store the file in IFS or use the Operations Navigator. It is important to convert the character code page between ASCII to EBCDIC.

3. From the navigation pane, expand **Manage Certificates**, and click **Import certificate**.

4. Select **Certificate Authority (CA)**, and click **Continue** in the window shown in Figure 29.

5. Insert the fully qualified name of the ASCII file containing the CA certificate as shown in Figure 30, and click **Continue**.

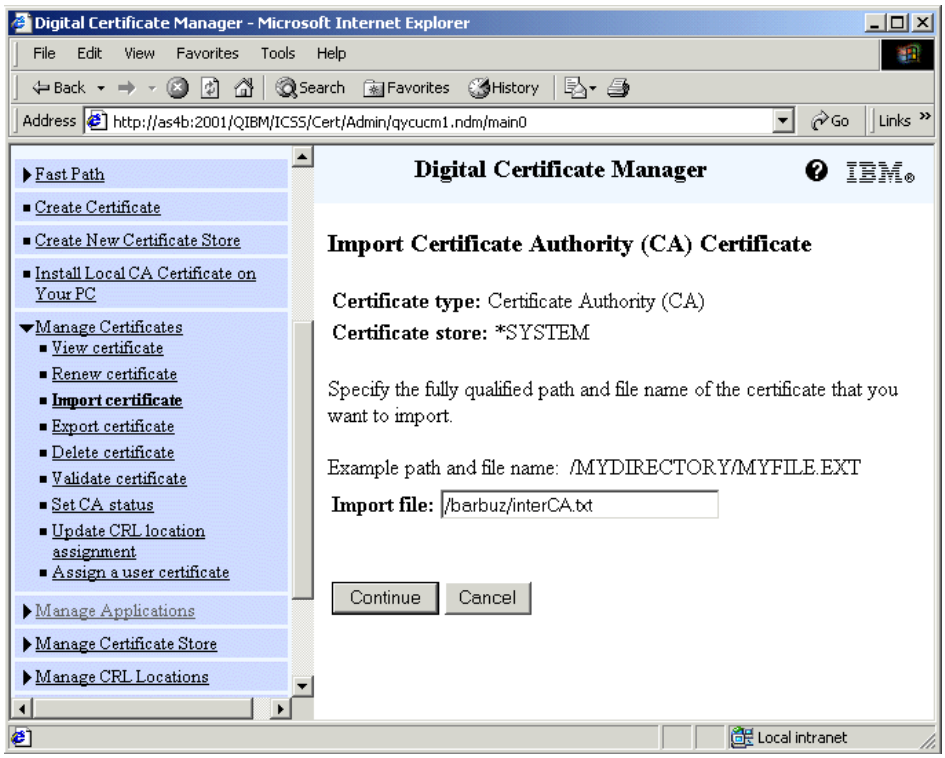


Figure 30. Import CA Certificate

6. Insert the CA certificate label as shown in Figure 31 on page 60.

Enter a name for the CA certificate that you are importing. That will be the unique identifier of the CA certificate in the certificate store, so you must specify a unique label.

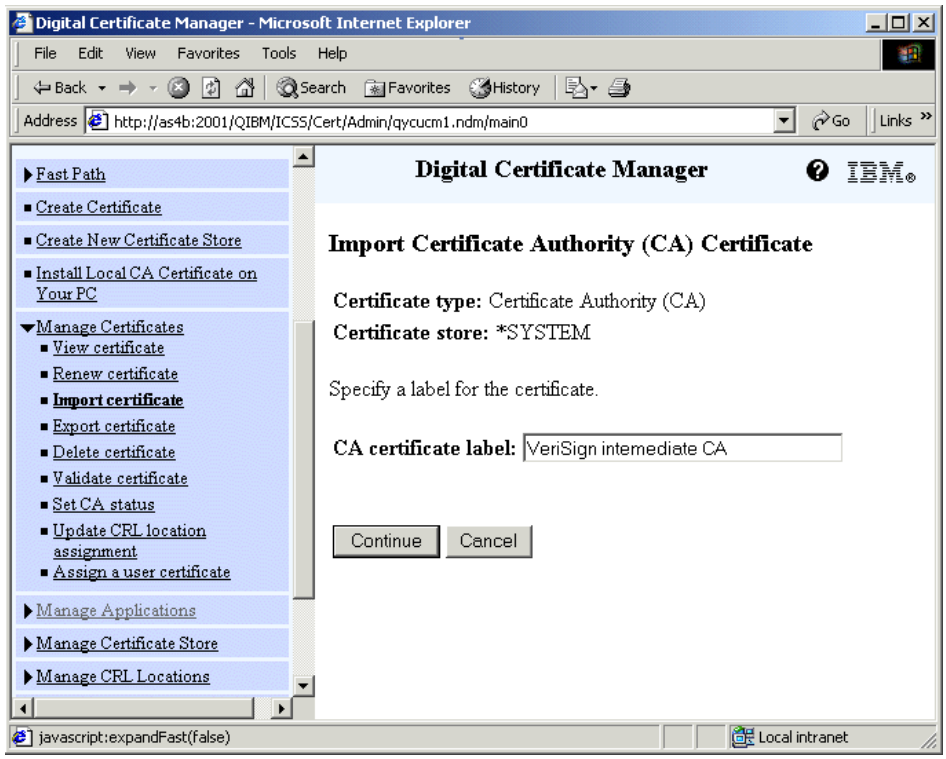


Figure 31. Specifying a CA certificate label

7. Click **Continue**.
8. Click **OK** in the completion window that appears.

   You can view this certificate in the Management Certificate task.

Each time you do not sign your server or client certificate with your local CA, you have to import it using DCM. It could happen when:

- You request a certificate from a well-known CA.

  Then you have to import only the digitally signed certificate, which does not contain the private key in the file. It usually uses the Base64 format to encode this certificate. The DCM checks whether a certificate request exists in the request database for the certificate to be imported. If the answer is positive, it checks whether the CA that issued the certificate is

present in the CAs list as trusted. Refer to 2.2.1, “Requesting a certificate from a local CA” on page 34, to see all the steps.

- You request a certificate from a PKIX CA or when you decide to use a certificate exported from another system.

  These situations are similar. In both cases, another CA generates for you both the pair of private and public keys and the digital certificate. The difference is how the two kinds of CAs handle the public key of the issued certificate. The PKIX CA holds a copy of the digitally signed certificate. But for the DCM that has to import the certificate, there are no differences. In V5R1, DCM supports and automatically detects the PKCS#7, the PKCS#12 V1 and V3, and the C3, which is an IBM internal format used with V4R3. The C3 format does not have an external standard reference.

Complete the following steps to import the certificate:

1. Obtain the server or client certificate from your CA.

   Usually in PKCS#7 or in PKCS#12 DER format, this is encoded to protect the private key. Therefore, the CA should also provide the password used to encrypt the file.

2. Transfer the PC file to an IFS directory on your AS/400.

   You can use FTP in binary format or map a directory to your AS/400 to store the file in IFS or use the Operations Navigator.

3. From the DCM navigation pane, expand **Manage Certificates**, and click **Import certificate**.

4. Select **Server or client**, and click **Continue**. See the window shown in Figure 29 on page 58.

5. Insert the fully qualified name of the file (stored in IFS) containing the certificate as shown in Figure 32 on page 62, and click **Continue**.

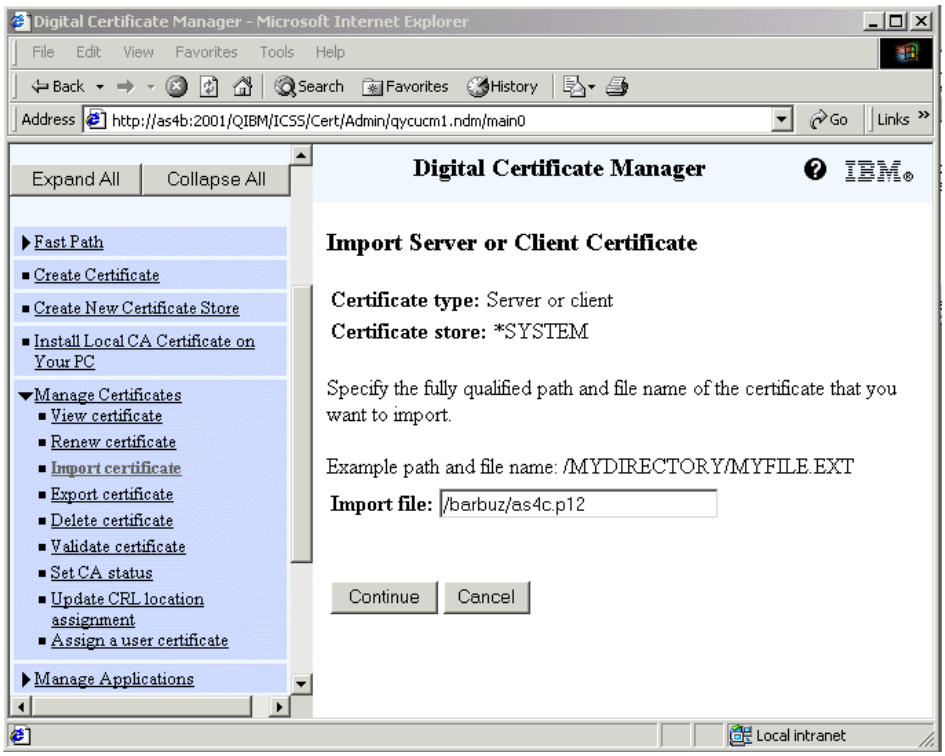


Figure 32. Import Server or Client Certificate task

6. Insert the password as shown in Figure 33, and click **Continue**.

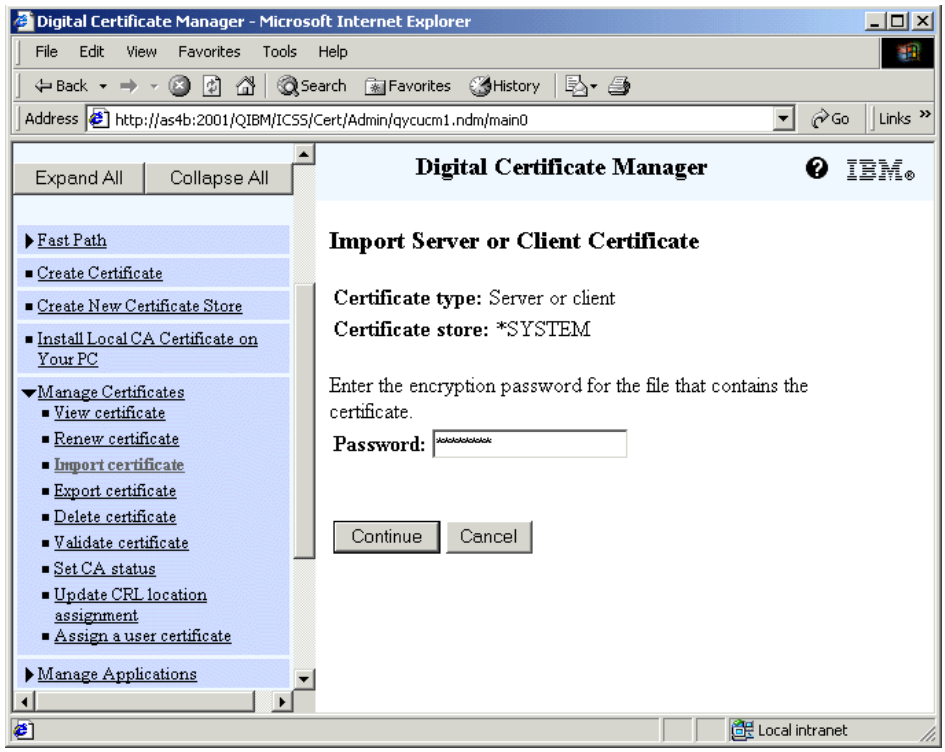


Figure 33. Inserting the password

Enter the password provided by the issuer of the certificate to decrypt the file.

7. Click **OK** on the completion window that appears.

   You can view this certificate in the Management Certificate task.

Here are some considerations:

- DCM does not give an error if the CA that issued the certificate is not present in the CAs list, because it can retrieve the CA certificate from the PKCS#12 encoded file. Then, the CA has the distinguished name label that is not immediately intelligible as shown in Figure 34 on page 64. We recommend you restore the CA certificate, if it is available, before you restore the server or client certificate.

	Certificate Authority (CA)	Status
◉	CN=loacalCA,OU=itso,O=ibm,L=raleigh,ST=nca,C=us	Enabled
○	AS4A Local CA	Enabled
○	SwissSign ID class 1	Enabled

Figure 34. Display CAs list

- When importing a certificate from a PKCS#12 file, DCM creates a certificate label based on an attribute called Friendly Name. This attribute is also stored in the PKCS#12 file. If the attribute has not been properly set, you see a label that appears to be out of the ordinary as shown in Figure 35. If you have more than one certificate with that kind of label, it will be more complicated to identify which certificate you want to manage. In this case, when the CA used for signing the certificate is a local one in your private network, and you do not need it to operate as a PKIX CA, if you choose, we recommend that you send a certificate request to the CA. And then import only the signed digital certificate.

	Certificate	Common name
◉	AS4B Local Cert (no extension)	as4b.itso.ral.ibm.com
○	61d566651970655cd8b556e369bc7745_254a15c0-637c-442d-b0cf-7a2cd2f843e3	as4c

Figure 35. Display server or client certificates list

- It is sometimes useful to manage the entire key database instead of importing one certificate, if DCM provides this opportunity. You can transport the entire key database (file.KDB) from another platform and open it in DCM by selecting *other as the certificate store and providing the fully qualified file name and its related password.

2.3.1.2 Exporting certificates

You can use your iSeries or AS/400 system as the local Certificate Authority. DCM can create server or client certificates for another system, but the AS/400 local CA cannot sign a certificate request received from another system. Therefore, it has to create the public/private key pair and the digital certificate. Then DCM allows you to export them to the target system.

DCM can also export its own CA certificate as shown in Figure 36. In V5R1, you can choose to export the selected type of certificate from the current

certificate store into another certificate store of your system or to a file that you can use on another system. When you export a certificate into another certificate store, you do not need to import it into the target store. The certificate will be immediately available in the certificate store selected as the target one. In this section,we place the export into a file option to focus on the interoperability problem between the AS/400 system and the other platform.

When you want to export a CA certificate, perform the following steps:

1. From the DCM navigation pane, expand **Manage Certificates**, and click **Export certificate**.
2. Select **Certificate Authority (CA)** as the type of certificate to be exported, and click **Continue** on the window as shown in Figure 36.

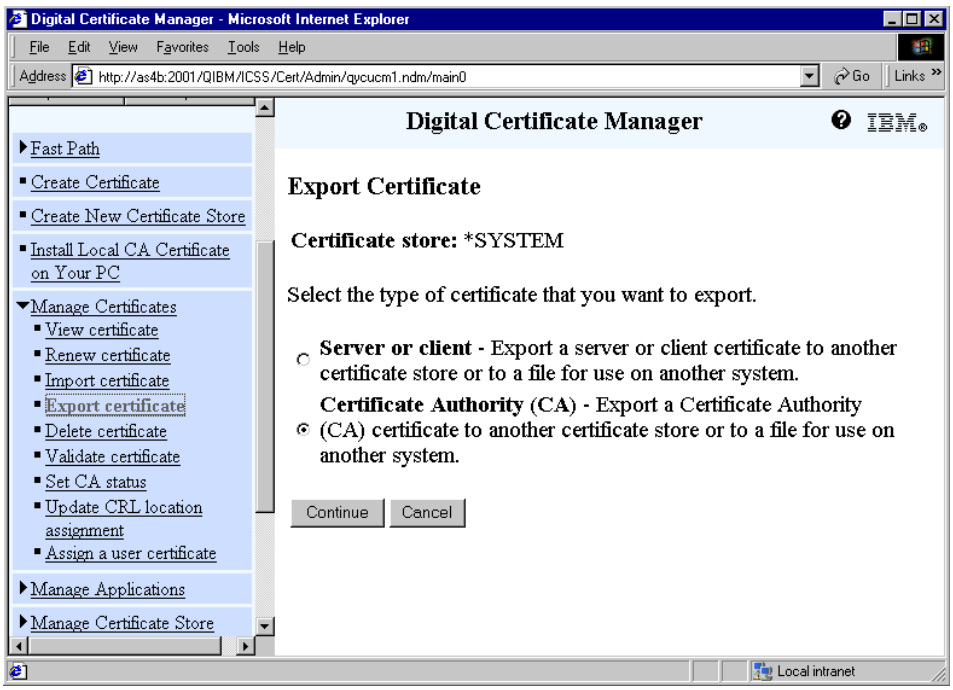


Figure 36. Export certificate task

The list of the CAs that are available in the selected certificate store appears.

The window shown in Figure 37 on page 66 appears. The DCM creates a file containing the CA certificate encoded with the privacy enhanced message (PEM) format.

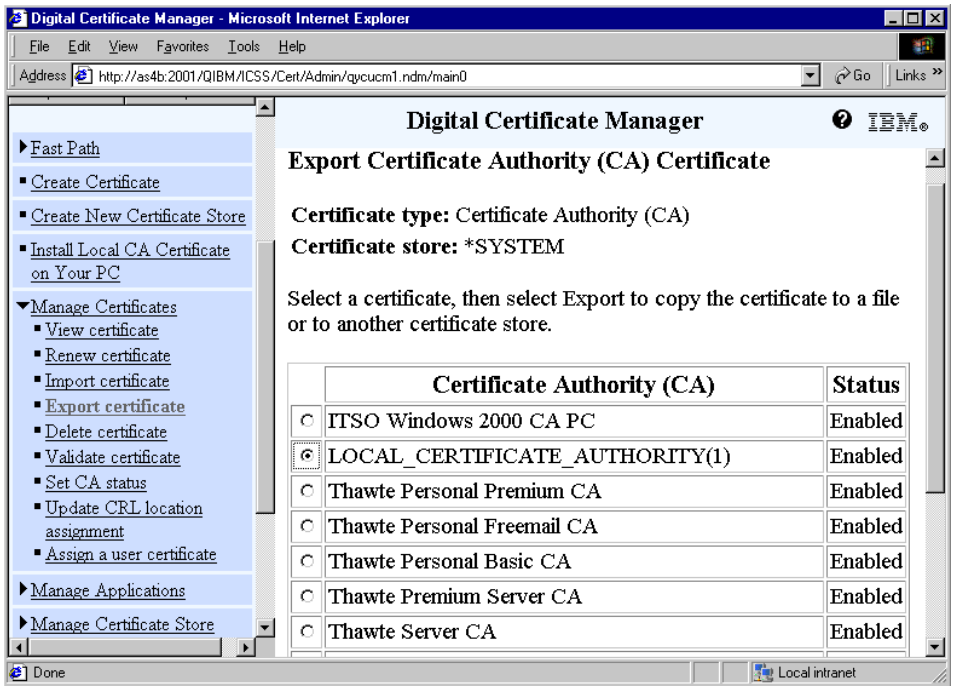


Figure 37. Export Certificate Authority (CA) Certificate window

3. Select a certificate to be exported, and click **Export**. You export only the CA digital certificate with its public key. It appears in the window shown in Figure 38 where you can choose how you want to export the CA certificate.

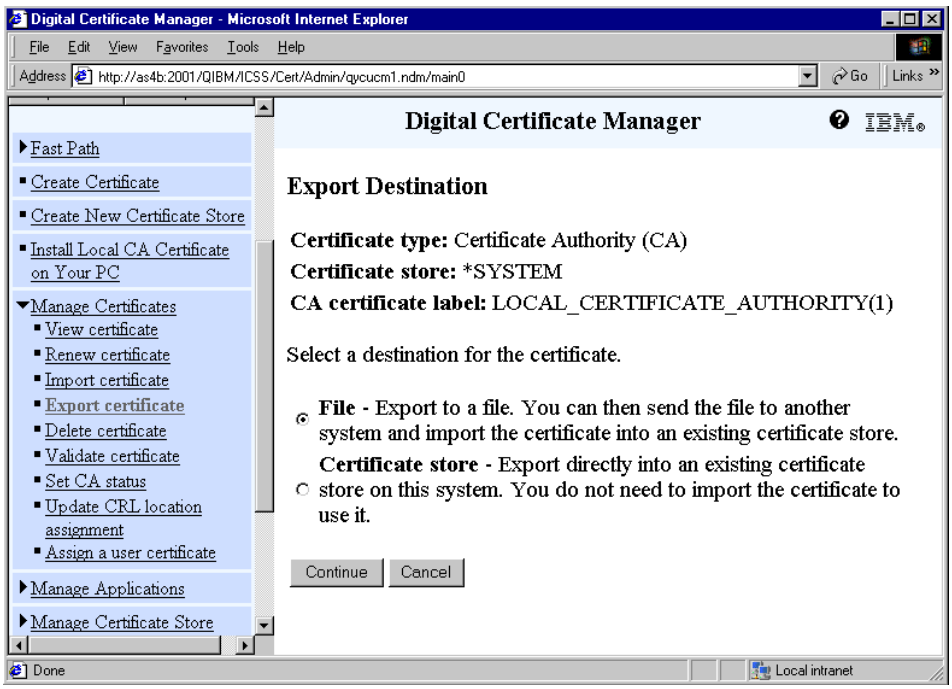


Figure 38. Selecting an export destination

4. Select **File**, and click **Continue**. A window like the example in Figure 39 on page 68 appears.

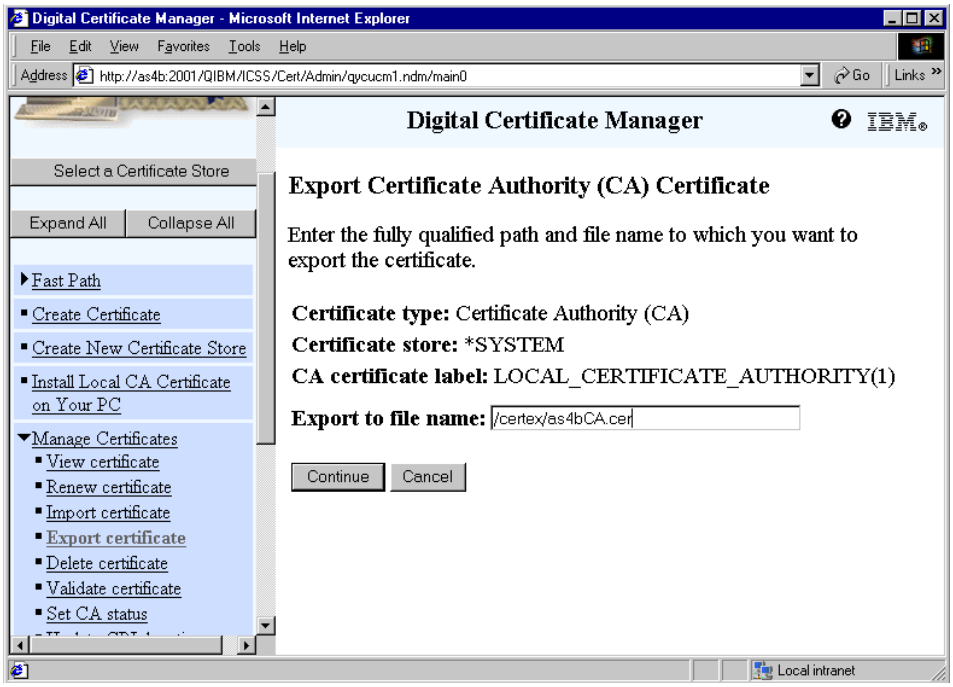


Figure 39. Exporting to a file name

5. Specify the fully qualified file name for the file that will contain the exported CA certificate, and click **Continue**.

   Enter the fully qualified path and file name that you want to use for the exported CA certificate. In the path name, you must include the leading slash. You can use any extensions for the file except an extension of .kdb. For example, it could be */mydirectory/myexportfile.txt*. In V5R1 and V4R5, the file is created with an EBCDIC code page.

6. Click **OK** in the completion window that appears.

You can now send the file into another system. Once it is on the target system, import it. Be aware that this is a text file, so you have to convert it from EBCDIC to ASCII. Transferring the file with FTP ASCII mode type is the recommended way.

Similarly if you want to export a server or client certificate, you have to follow these steps:

1. From the DCM navigation pane, expand **Manage Certificates**, and click **Export certificate**.

2. Select **Server or client**, and click **Continue** in the window shown in Figure 36 on page 65.

   The list of the server or client certificates that are available in the selected certificate store appear.

3. Select a server or client certificate, and click **Export**.

   You export both the private key and the digital certificate. You cannot export a certificate if the certificate uses a cryptographic coprocessor to store or to encrypt the certificate's private key.

   It appears in the window shown in Figure 40 where you can choose how you want to export the certificate.

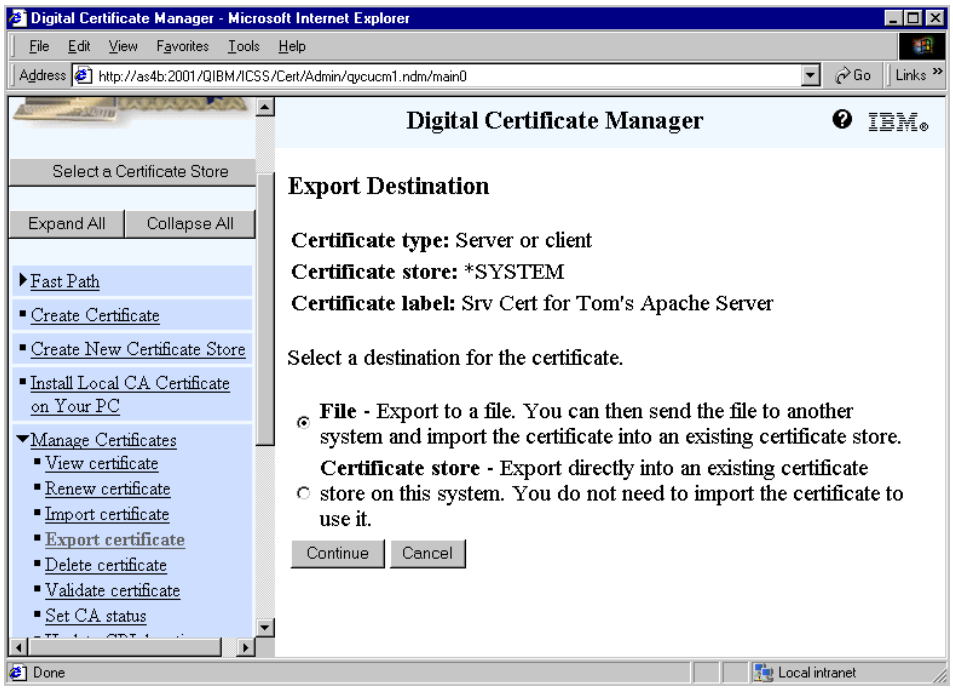


Figure 40. Selecting an export destination

4. Select **File**, and click **Continue**.

   The window shown in Figure 41 on page 70 appears. The DCM creates a file containing the server or client certificate and its issuer chain encoded with the PKCS#12 DER format. When exporting certificates into another certificate store, you need to provide the name of the target certificate store and its password.

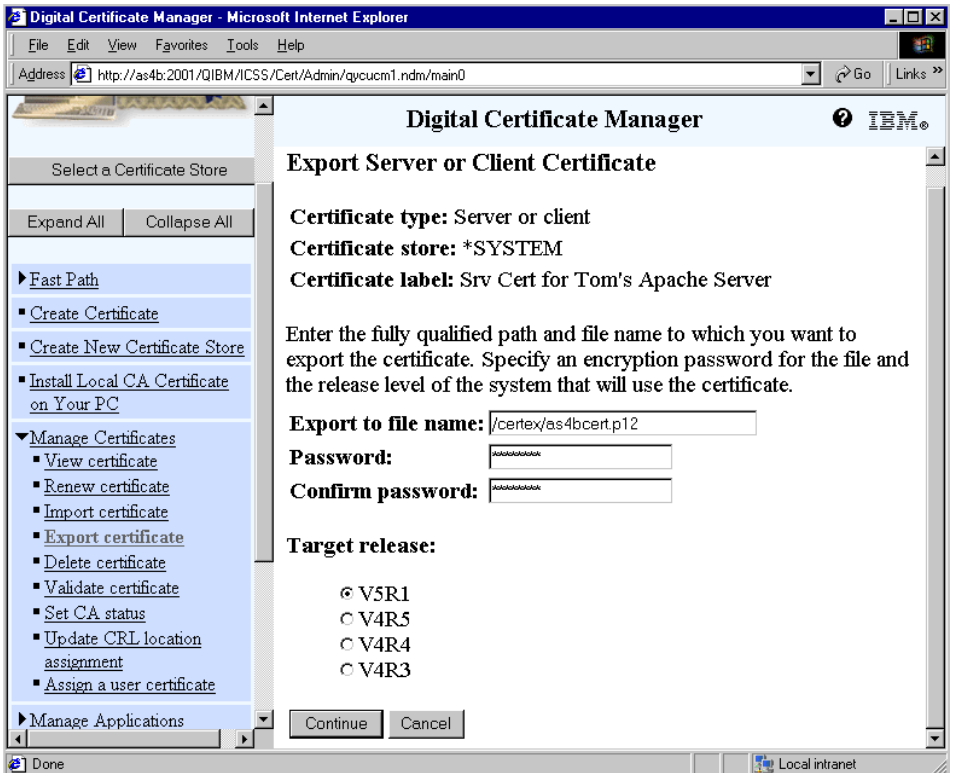


Figure 41. Export server and client certificate in the file name

5. Fill in the requested fields shown in Figure 41 to specify the file name and the format to use. Click **Continue**.

   In this window, you have to:

   a. Enter the fully qualified path and file name that you want to use for exporting the certificate. In the path name, you must include the leading slash. You can use any extension for the file except an extension of .kdb. For example, it could also be */mydirectory/myexportfile.txt.*

   b. Enter the password that you want to use to protect the new file from unauthorized access two times to confirm it. The password is case sensitive.

   c. Select the release level of the AS/400 for which you are creating the export file. If there are different platforms, the release level that you select determines the format that DCM uses to copy the certificate into

the export file. This ensures that the certificate is compatible with the system that will use the certificate. When you select:

- V5R1, you obtain PKCS#12 Version 3 encoded certificates.
- V4R5 and V4R4, you obtain PKCS#12 Version 1 encoded certificates.
- V4R3, you obtain C3 (IBM Internal format) encoded certificates.

Then, a completion message appears.

You can now transport the file into another system. Once it is on the target system, import it. Remember that, in this case, the file is in binary format. Transfer the image of the file via FTP with a binary mode type. You can also drag and drop it. It is important that you do not apply any translation.

2.4 Managing applications

Manage Applications tasks are available in the *SYSTEM and *OBJECTSIGNING certificate stores only. Figure 42 on page 72 shows you the window that appears if you click **Manage Applications** from the DCM main task in the *SYSTEM certificate store.

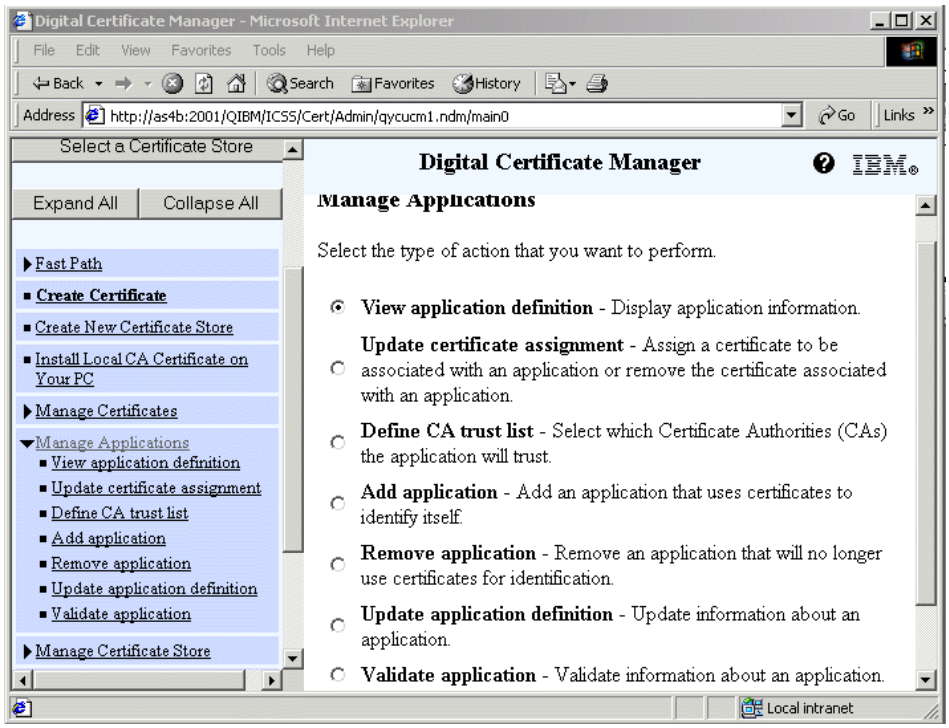


Figure 42. Manage Applications task

In this task you can view, add, remove or update the information for an application. You can also change the certificate assignment for an application for server or client authentication or assign which CAs an application can trust during the client authentication SSL phase. You can also validate information for an application definition that could be useful to prevent certificate problems for the application when it is performing a function that requires certificates.

The following section covers the main management function for an SSL-enabled application that you can handle in the *SYSTEM certificate store.

2.4.1 Defining applications

Prior to V5R1, the only way to add an application to the DCM was to use an API (such as QSYSRGAP, QsyRegisterAppForCertUse). Now you can add applications using the DCM. Using a graphical interface is much easier than using the API. You can add, remove, or update both server or client SSL

applications directly in the DCM task. For more information, see 6.5, “Adding an application description to the Digital Certificate Manager” on page 348.

You cannot add or remove standard AS/400 applications. DCM allows you only to update its application definition fields that it already enabled as personalized. Regardless of whether you want to require a client authentication on your FTP server, you have to perform the following steps:

1. From the DCM navigation pane, expand **Manage Applications**.
2. Click **Update application definition**. A window appears like the example shown in Figure 43.

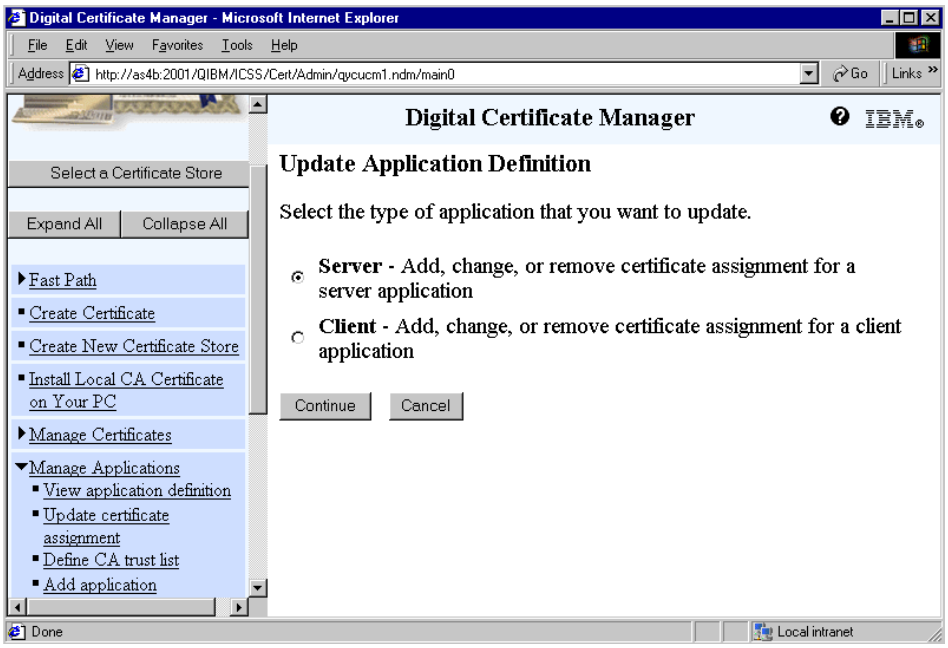


Figure 43. Update Application Definition: Application type selection

3. Select which type of application, **Server** or **Client**, you want to update, and click **Continue**.

   Since we decided to update the FTP server application definitions in this example, we selected server applications. The window shown in Figure 44 on page 74 appears.

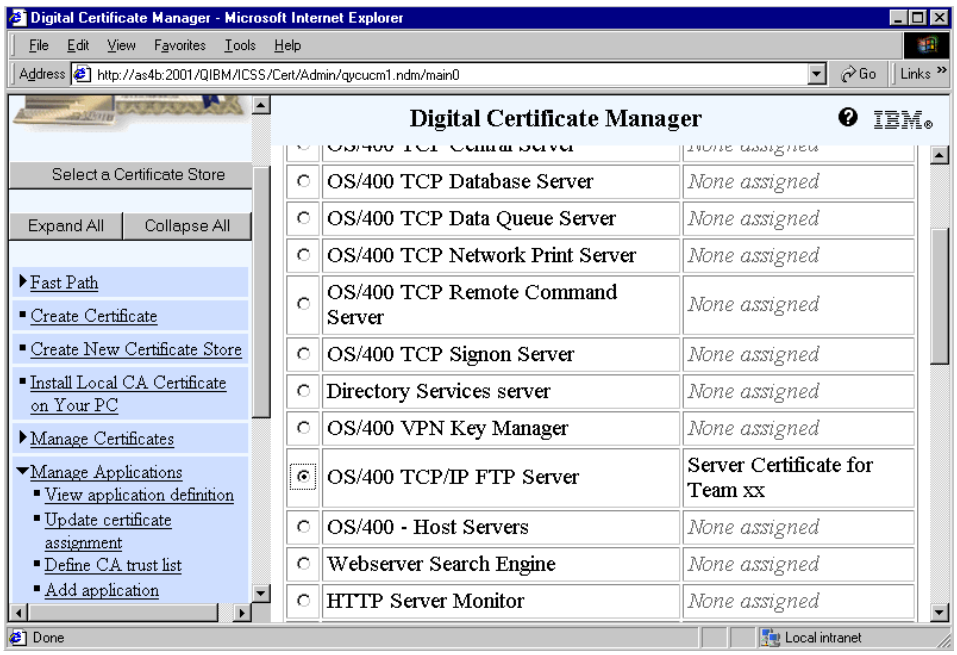


Figure 44. Selecting the application that you want to update

4. Select the application for which you want update the definitions from the list of applications. Click **Update Application Definition**.

   For this example, we selected the OS/400 TCP/IP FTP Server. The window shown in Figure 45 appears.

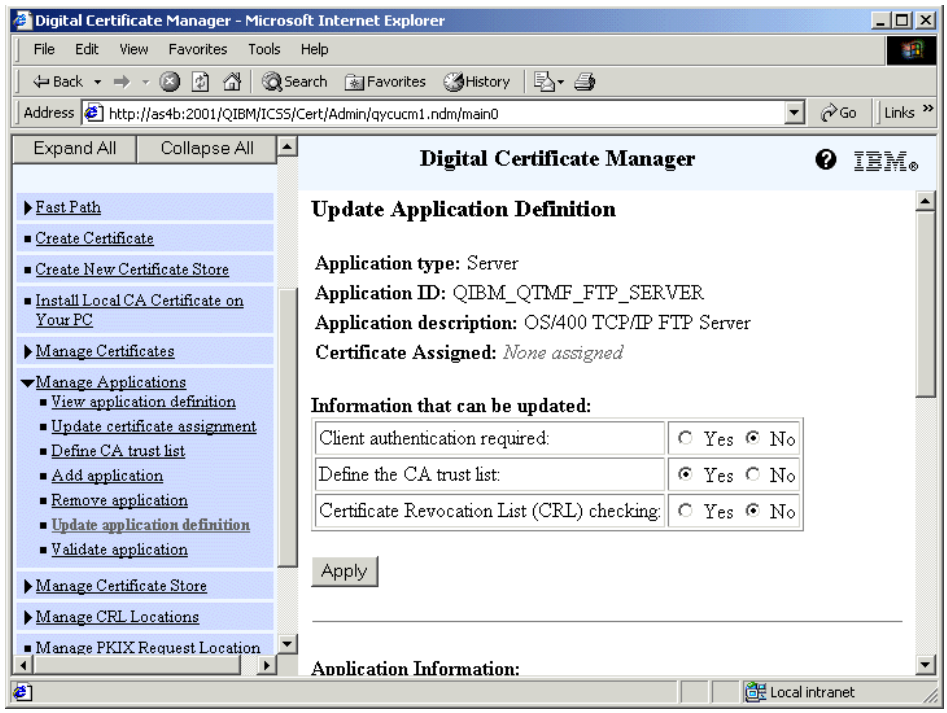


Figure 45. Updating the definition for the selected application

5. Change the information in the application definition, and click **Apply**.

   Only the application definition fields that you can change are displayed. The items that are displayed vary among application types. The following options show the entire list of information that you could eventually personalize in the standard applications:

   - **Client authentication required**: Select a value to indicate whether the application requires client authentication before allowing user access to the application.
   - **Certificate Revocation List (CRL) checking**: Select a value to indicate whether SSL checks a Certificate Revocation List (CRL) location when validating the assigned certificate for the application when establishing an SSL session.
   - **Define the CA trust list**: Select a value to determine if the application should refer to the list of trusted CAs or if the application should trust all CAs with a status of enabled in the *SYSTEM certificate store.

   A message is displayed at the top of the page to indicate the status of the change request.

6. Click **Cancel** to go back to the list of selectable applications that you can update.

2.4.2 Assigning CA trust

Client authentication during an SSL handshake session allows an application to determine whether to accept a certificate as valid proof of identity of the client. One of the criteria that an application uses for authenticating a certificate is whether the application trusts the CA that issued the certificate. DCM only allows you to define a CA trust list for client applications or server applications that support client authentication, with the exception of HTTP servers.

For both the original server and servers powered by Apache, you can also define a CA trust list even if they are registered with client authentication that is not supported. If the definition for one of those applications specifies that the application use a CA trust list, you must define the list before the application can perform certificate client authentication successfully. This ensures that the application can validate only those certificates from CAs that you specify as trusted. If users or a client application present a certificate from a CA that is not specified as trusted in the CA trust list, the application will not accept it as a basis for valid authentication.

To personalize the CA trusted list for a specific application, follow these steps:

1. From the DCM navigation pane, expand **Manage Applications**.
2. Click **Define CA trust list**.

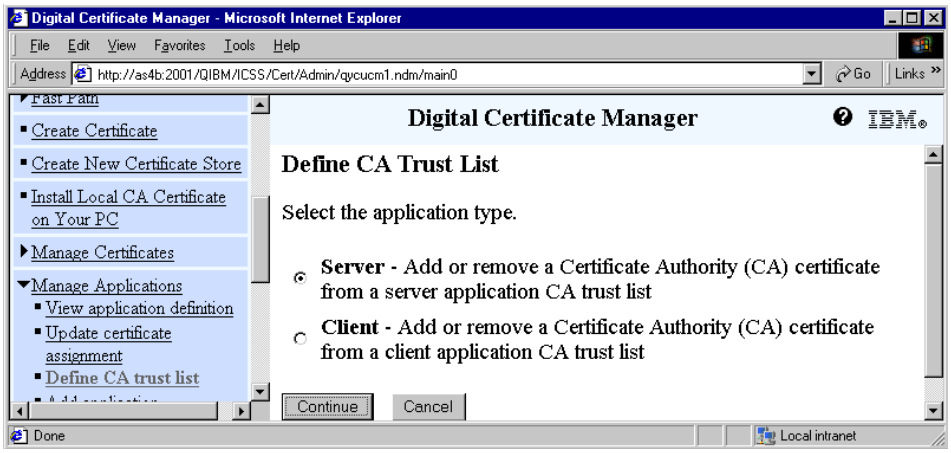


Figure 46. Define CA Trust List

3. Select for which type of application, **Server** or **Client**, you want to define the CA trust list. Click **Continue**.

   For this example, we chose to define the CA list for the server application. The window shown in Figure 47 appears.

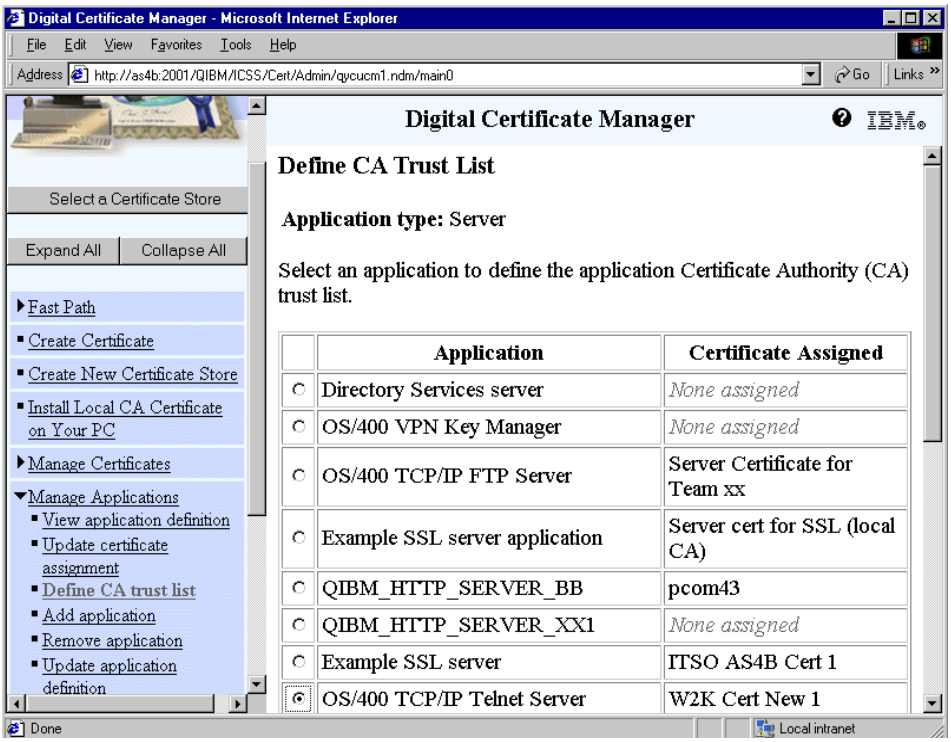


Figure 47. Define CA Trust List

4. Select an application from the list, and click **Define CA Trust List**.

   In this example, we selected the Telnet server application ID. A list of CA certificates appears as shown in Figure 48 on page 78. This is the list of CAs that are in enabled status in this certificate store. You will use this list to define the trust list for the selected application.

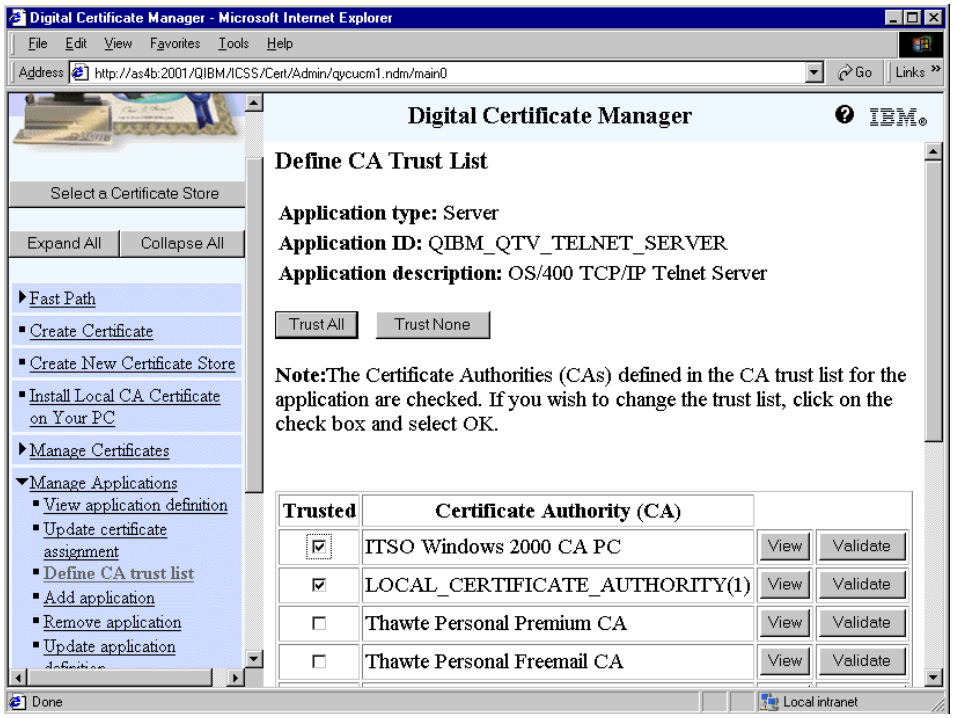


Figure 48. Selecting the CAs trusted in the list

5. Check the CAs that the application should trust, and click **OK**.

   You should see a checkmark next to the CAs that are already selected as trusted in this list. You can define more than one CA as trusted for the application by selecting multiple CAs from the list. You can also remove some CAs from the trust list by unchecking them. After you submit your changes, a message is displayed at the top of the page to indicate whether the changes were successful, or to describe errors if the changes failed.

   The Trust All and Trust None buttons are also available. Even if it is not strictly necessary, you should not trust all the CAs that are enabled, to ensure that the application can validate only those certificates from CAs that you specify as trusted.

6. Click **Done** to go back to the list of selectable applications shown in Figure 47.

2.4.3 Assigning certificates

For an application to be SSL-enabled, you have to assign a certificate to provide “proof of identity” to your server or to your client applications. When you assign a certificate to a server application, it uses this certificate during the SSL handshake to perform server authentication. In V5R1, there are also two client OS/400 standard applications, such as the Directory Service Client and Publishing, that can provide a client certificate when the server requires authentication. You must use DCM to assign a certificate to an application before the application can perform a secure function, such as establishing a SSL session or object signing. All applications must be registered in DCM prior to assigning a certificate. The AS/400 standard applications are automatically registered as secure applications in DCM. Some of these applications are not shown in the list of secured applications unless they are enabled for SSL.

The only exception is the OS/400 VPN Key Manager application. You do not need to assign a certificate to this application, since certificate selection is not handled by DCM. The certificate to be used for identifying the server is based on the server certificates in the *SYSTEM store that are issued by a CA trusted by the OS/400 VPN Key Manager as well as the VPN Internet Key Exchange policy configuration.

To assign a certificate to an application, or to change the certificate assignment for an application, follow these steps:

1. From the DCM navigation pane, expand **Manage Applications**.
2. Click **Update certificate assignment**.

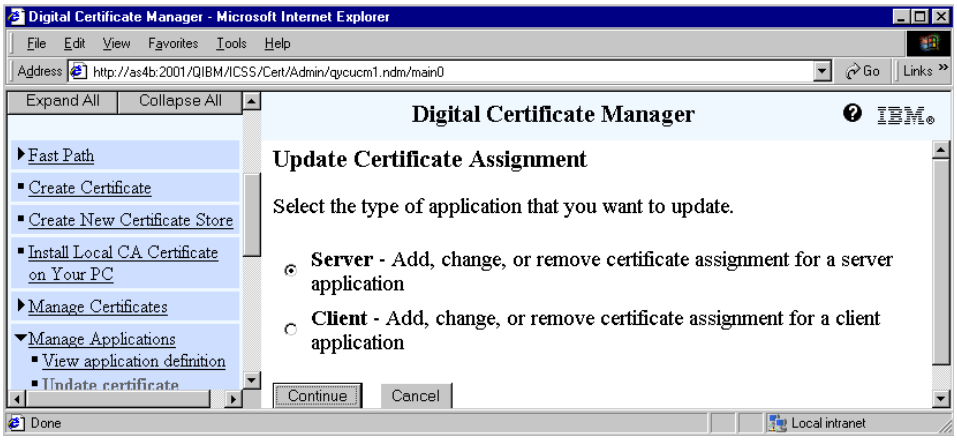


Figure 49. Update Certificate Assignment: Application type selection

3. Select for which type of application, **Server** or **Client**, you want to make a certificate assignment change. Click **Continue**.

   For this example, we selected to change the certificate assignment for the server applications. The window shown in Figure 50 appears.

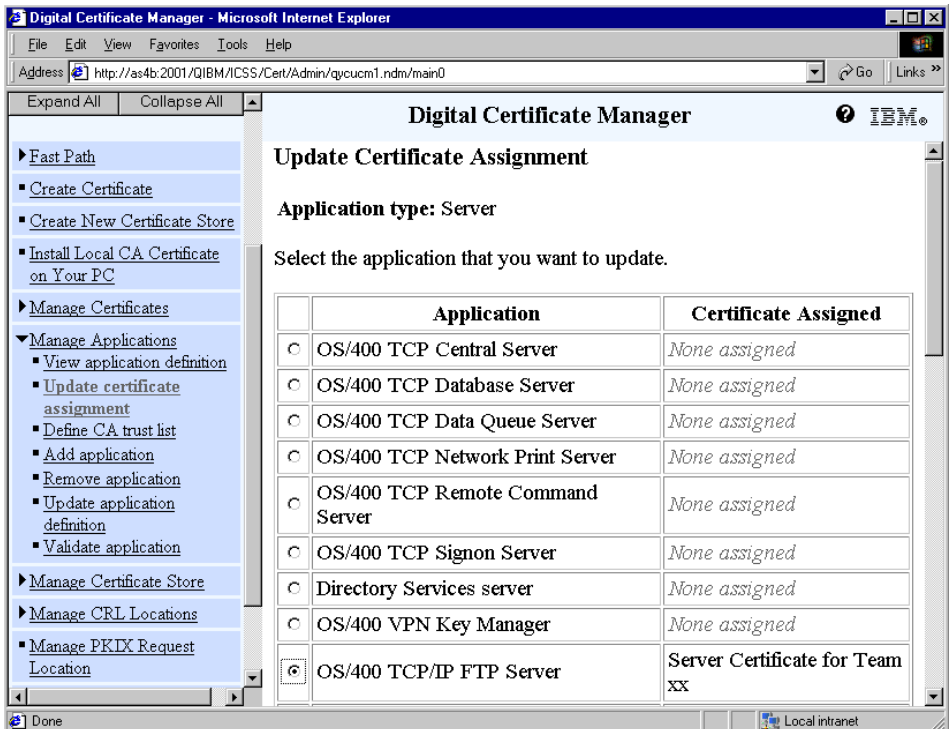


Figure 50. Selecting an application for which updating certificate assignment

4. Select an application from the list for which you want to change the certificate assignment. Click **Update Certificate Assignment**.

   Each entry in the list provides the application name and the certificate label name of the current certificate assignment for the application. For this example, we selected the FTP server application. The window shown in Figure 51 appears.

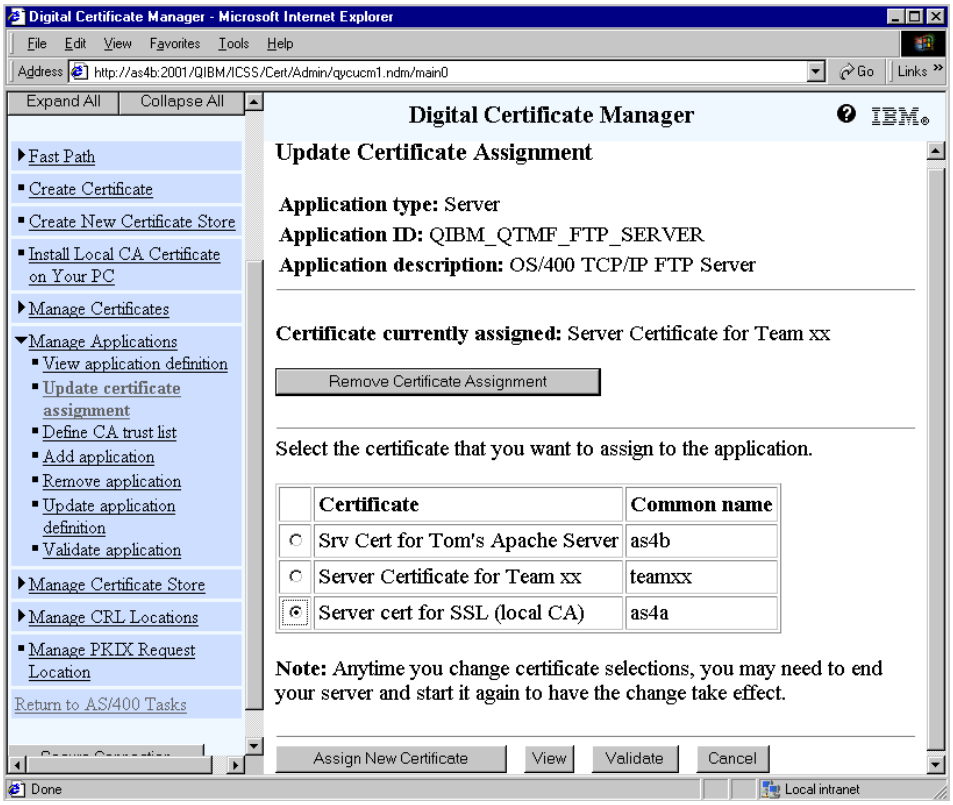


Figure 51. Update Certificate Assignment for the selected application

5. Select one of the server certificates available in the list, and click **Assign New Certificate**.

   You can also remove the current certificate assignment for the application. In both cases, a message is displayed at the top of the page to confirm the certificate assignment change.

   Notice that when you change a certificate assignment for an application, you may need to end your application and start it again to have the change take effect.

6. Click **Cancel** to go back to the list of the selectable applications.

2.5 Exploiting the Certificate Revocation List support

Before discussing how CRL processing is implemented in OS/400, it is helpful to have an overview of what CRLs are in general.

Certificates have an expiration date, which means they must be periodically renewed. This limits the time period in which a compromised certificate could be used. If a private key were to be compromised, the associated certificate can no longer be trusted.

A certificate can be invalidated by placing it on a Certification Revocation List or CRL (refer to RFC 2459). CRLs are lists of certificates issued by a particular CA that have been revoked for some reason. Many different reasons exist for a certificate to be revoked. For example, the private key associated with the certificate may has been compromised, data in the certificate's distinguished name may have changed, or the person represented by the certificate may no longer work for the company.

You achieve a higher level of security by checking whether the certificate that you are using has been placed in a CRL during the validation or the authentication of the certificate itself. CRLs contain information about the issuer, the time/date when the CRL has been issued, the time/date when the next CRL is scheduled to be issued, and the list of certificate serial numbers that have been revoked by this issuer. The CA who issues the CRL can also decide to include a reason code for the revocation. The CA that issued the certificate is responsible for issuing CRLs.

Sometimes when the list contains the certificates of revoked Intermediate CAs, it is called the *Authorities Revocation List* (ARL). On the iSeries and AS/400 server, both the CRL and ARL are referred to as CRL and are handled in the same way. Since the CA is not involved at the time this certificate validation is being performed (for example, during signature validation time, etc.), the CAs periodically update their CRLs and make them available for others by publishing them in Lightweight Directory Access Protocol (LDAP) directories. Servers or clients that accept certificates as proof of identity should verify that the certificate has not been revoked by periodically obtaining the latest CRL from a directory server or verifying the authenticity online by accessing the CA directory containing the CRL.

DCM allows you to define and manage CRL location information to ensure more stringent authentication for certificates that you use or you accept from others. A CRL location definition describes the location of, and accesses the information for, the Lightweight Directory Access Protocol (LDAP) server that stores the CRL.

You can use DCM to validate individual certificates, or to validate an application and the certificate that it uses to ensure its authenticity. One of the steps associated with validating a certificate is to see if the certificate being processed has been placed in a CRL. As a certificate is validated, there is a check to see if the extension called “CRL distribution points” exists. This extension provides the search parameters for specifying what CRL or CRLs the LDAP client should retrieve from the LDAP server.

Note: The CRL distribution point extension does not provide information on where the LDAP server is. It provides information on where the CRL(s) pertaining to this certificate are located (within the X.500 directory structure). If no CRL distribution point extension exists, DCM falls back to specifying the issuer distinguished name (DN) as the LDAP search parameter for the CRLs it wants to retrieve. CRL distribution points provide a means of subsetting CRLs provided by a CA. For more information about the structure of a CRL query, refer to Appendix F, “Publishing a CRL to an OS/400 LDAP server” on page 451. Although the default when validating a certificate is to use CRLs, if there is no LDAP server designated, CRL checking is not performed. However, the certificate is still subject to the normal validation process.

The validation process is invoked manually, selecting the related option in the Manage Certificates and Applications tasks, or automatically during the association of a certificate to an application. In addition, applications that support the use of certificates for client authentication can perform CRL processing to ensure more stringent authentication for certificates that they accept as valid proof of identity.

In V5R1, the applications and processes that may perform CRL processing for certificate authentication are:

- Virtual private networking (VPN) Internet Key Exchange (IKE) server
- The object signing and signature verification process
- Secure Sockets Layer (SSL) enabled applications

This implementation increases the security checks of your application, but be aware that there could be a performance issue. Each time the authentication process involves a CA with a CRL location assigned, the application queries the CRL LDAP server.

In V5R1, is not possible to cache the CRL list, so for each validation, a new connection is established. This could cause a delay for a user during network congestion or an abend for the authentication phase when the application cannot reach the server.

If a certificate is found on a CRL, it is not considered an exception error as such. However, it is returned as an error to DCM. A message to the effect that the certificate has been revoked is then issued to the user. For other LDAP errors (for example, the server is not reachable or the directory searched is not on the LDAP server), a message that there is no CRL entry is issued and the certificate cannot be validated. It is worthwhile mentioning that a certificate listed in an expired CRL is considered valid as long as the signature and validity period of the certificate are still valid.

Configuration and maintenance of the information required for our Certificate Management support to communicate with LDAP servers for retrieving CRLs consists of two separate parts:

- Configuration of the LDAP-specific information (host name, IP address, user name, and password). Refer to 2.5.1, “Managing CRL locations” on page 84.
- Configuration of the association between a particular CA certificate and the LDAP server that handles publishing that CA's CRLs. Refer to 2.5.2, “Assigning CRL locations to CA certificates” on page 87.

After these steps, the DCM can perform a CRL checking process during the validation task.

To enable an application (that is, SSL Sockets application) to use a defined CRL as part of the certificate validation process, you have to enable the CRL checking process in the DCM application definition. Refer to 2.4, “Managing applications” on page 71.

2.5.1 Managing CRL locations

If a certificate has to be validated using CRLs, then its issuer or issuer's issuer should be configured with a CRL location. The CRL location is the LDAP server where the issuer publishes its CRLs and should be published under its distinguished name.

In DCM, to create or to manage a CRL location, click the **Manage CRL Locations** task. This task is always available regardless of the certificate store you select. You can use this task to choose the type of management task that you want to perform for Certificate Revocation List (CRL) locations.

As shown in Figure 52, you can view, add, update, and remove a CRL location. A CRL location definition describes the location of, and access information for, the Lightweight Directory Access Protocol (LDAP) server that stores the CRL. The DCM allows you to specify more than one single LDAP server in this task.

The association between an LDAP server and a CA is configured later inside the certificate store in the Manage Certificates task. Applications that perform certificate authentication can access the CRL for a specific CA to ensure that the CA has not revoked a specific certificate.

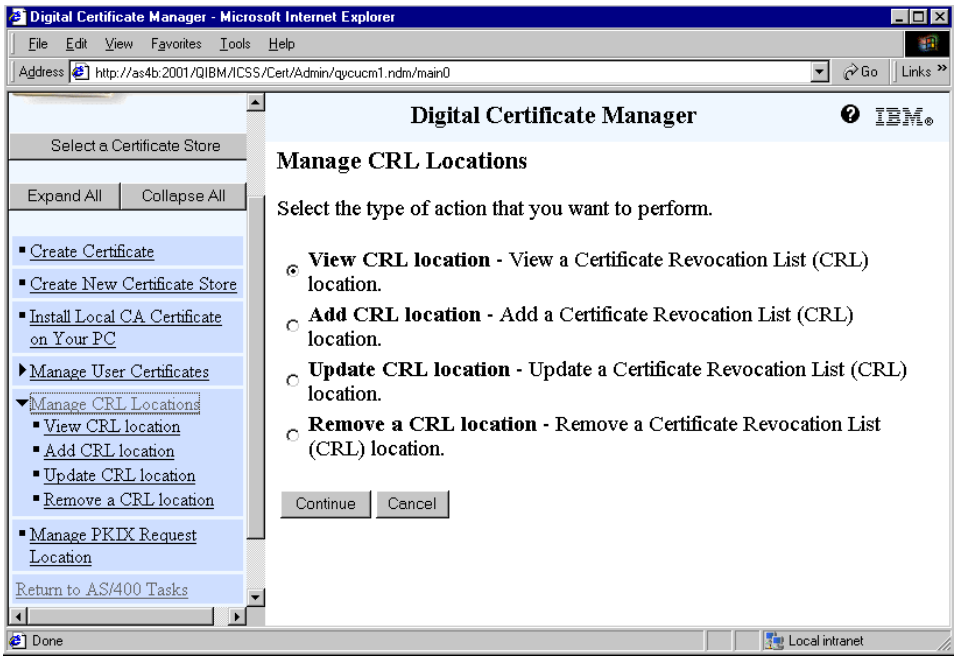


Figure 52. Manage CRL Locations task

To define a new Certificate Revocation List (CRL) location, you have to perform the following steps:

1. From the DCM navigation pane, expand **Manage CRL Locations**.
2. Click **Add CRL location**.

   You see the window shown in Figure 53 on page 86. All the information required in this window should be provided by the CA with whom you want to implement CRL checking.

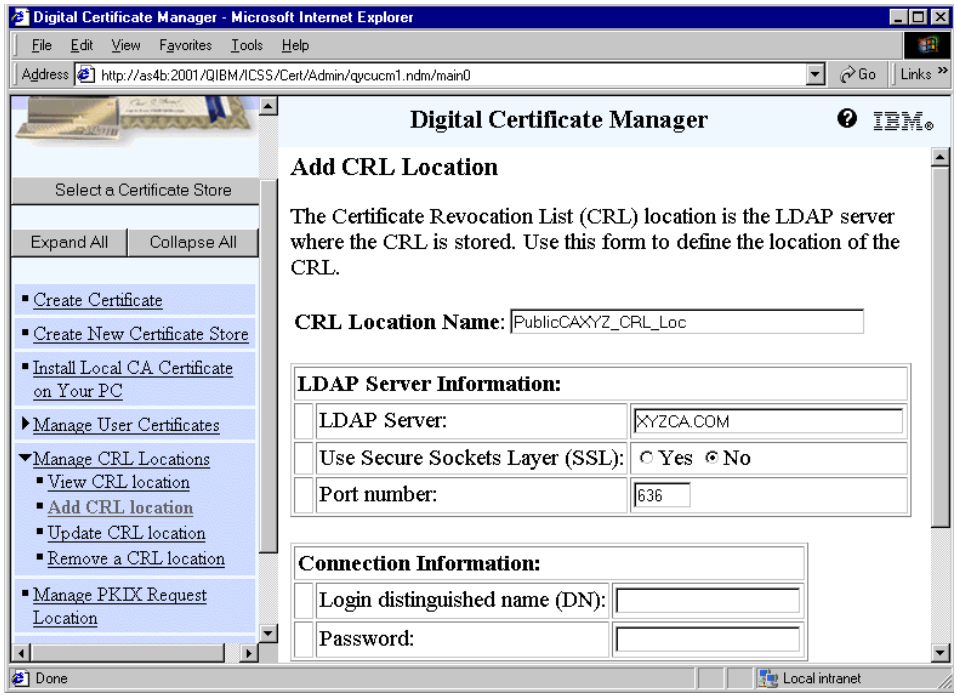


Figure 53. Add CRL Location task

3. Fill in the required parameters:
   - **CRL Location Name**: Enter a name for the CRL location. This is a label for this CRL location. Because this name uniquely identifies the CRL, you must specify a unique name. You can use from 1 to 100 alphanumeric characters for the name.
   - **LDAP Server Information**
     - In the LDAP Server field, enter the domain name of the LDAP server on which the CRL is located. This could be a DNS name, such as VeriSign.com, or an IP address.
     - In the Use Secure Sockets Layer (SSL) field, specify whether or not the CRL location uses Secure Sockets Layer (SSL) for secure access. In this particular case, it could be useful to enable the SSL to authenticate the LDAP server that is providing such critical security information. If you choose to use SSL, you will have to provide the secure port number of the LDAP server and will define a CA trust list for your LDAP client application in the *SYSTEM certificate store.

- In the Port number field, enter the port number of the LDAP server. The standard non-secure port number is 389 and the standard secure port number is 636.

- **Connection Information**
  - In the Login distinguished name (DN) field, enter the login distinguished name for the LDAP server.
  - In the Password field enter the password that is associated with the login distinguished name. This field does not display unless the value for the Retain Server Security Data (QRETSRVSEC) system value is '1'. A value of '1' for this system value means that the system can store decrypt able passwords. If the QRETSRVSEC value is '0', then this option is not available because the AS/400 cannot store passwords in clear text and DCM cannot access them. If you enter a password, you must also enter a login distinguished name.

To use an anonymous session to the LDAP server for CRL processing, leave both the Login distinguished name field and the Password field blank.

4. Click **OK** to confirm the parameters.

   A completion message appears. Then click **OK** to return to the previous task.

2.5.2 Assigning CRL locations to CA certificates

The CAs are responsible for maintaining their CRL location. On DCM, you can assign a CRL location to a particular CA. Multiple Certificate Authorities could refer to the same LDAP server for their CRL lists. DCM allows you to assign the same LDAP server label ID to multiple CAs, but you can assign only one LDAP server to a single CA at a time. Limiting the CRL search to a single LDAP server for any given certificate being validated limits the delay caused by this checking.

To assign a CRL location to a CA, complete these steps:

1. Open the certificate store.
2. From the DCM navigation pane, expand **Manage Certificates**.
3. Click **Update CRL location assignment**.

   You see a window like the example in Figure 54 on page 88.

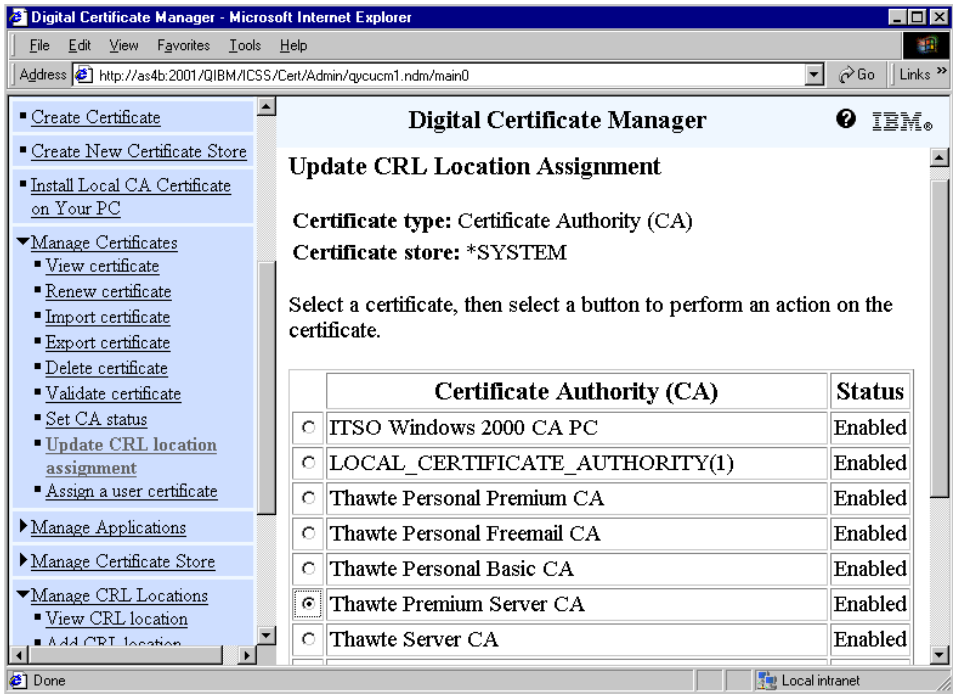


Figure 54. Choosing the CA for updating the CRL location assignment

4. Select the Certificate Authority (CA) certificate for which you want to update the Certificate Revocation List (CRL) location assignment. You can assign a CRL location to any CA certificate in the list of available CA certificates for the current store.
5. Click **Update CRL Location Assignment**.

   You see the window shown in Figure 55. Select the CRL location you want to assign, and click **Update Assignment**.

   In this window, you can also remove a CRL location that was previously assigned by clicking **Remove assignment**. The Remove assignment button is not shown until a CRL location is assigned.

   In both cases, you receive a completion message.

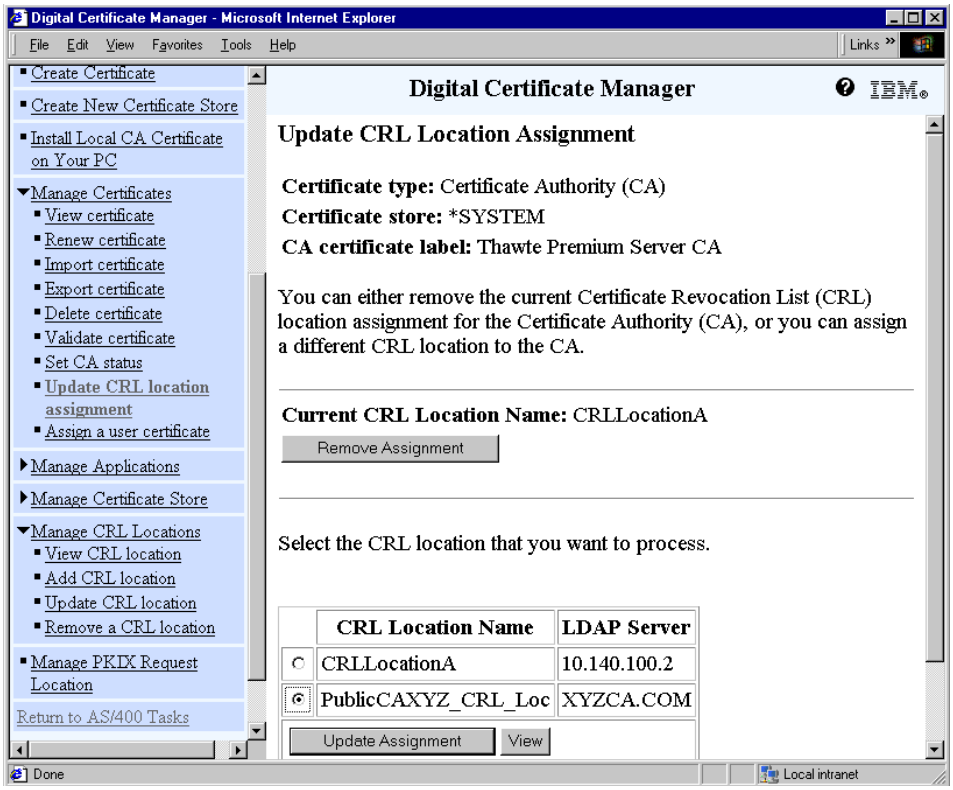


Figure 55. Update CRL Location Assignment

6. Click **Cancel** to return to the list of the CAs.
7. If you want to update another CRL location assignment, repeat the steps starting from step 4 for the other CA.

If you want to use CRL checking for one of those applications that allow it, you also have to modify the application definition. Refer to 2.4.1, “Defining applications” on page 72.

2.5.3 Performance considerations

If you consider using a CRL implementation to increase the security checks of your application, you also have to consider how this could impact performance. Keep in mind that each time the authentication process involves a CA with a CRL location assigned, the DCM will query the LDAP server specified in the CRL location, which could introduce delays in your application. CRLs are not cached in V5R1. If you consider that the CRL server is usually located on the Internet, you will see that the amount of these

delays introduced by CRL checking depends on a lot of factors. Some factors are related to your Internet access: line speed, whether it is dial-on demand, whether you have to pass through a firewall, and eventually network congestions. These are just a few of the possible elements that could affect your performance.

On the other hand, some factors are related to the response times of the LDAP server itself. In addition, if a CRL is not found at the CRL location, validation of the certificate cannot be performed and an error is returned to your application. To avoid this, you should be sure that the CA has published CRLs to the location you specify and that you can reach the location through the network.

Most CAs give you the opportunity to download the entire CRL list. This allows you to eventually store the CRL in a local LDAP directory server. This gives you better answer times and more reliability of the network. The CRL lists have information about the time and date when the next CRL is scheduled to be released. Until that time, you can trust the CRL replication done on your LDAP server. Since there is no easy way to programmatically access the attribute of the date on which the next CRL is to be published, you have to obtain this information from the CA. For the fastest response times, you can configure an LDAP server on your iSeries or AS/400 system. Then both the LDAP server and client will be on the same machine. In this case, you would specify as the CRL location the loopback address (127.0.0.1) of the AS/400 system. This configuration also has the advantage that the LDAP query data traffic is handled internally on the system and will not use the local network at all. The entire process of setting up a local LDAP server, including publishing the CRL, is documented in Appendix F, “Publishing a CRL to an OS/400 LDAP server” on page 451.

2.6 Application programming interfaces

This section introduces OS/400 APIs that can be used to work with digital certificates. Digital Certificate Manager (DCM) centralizes management functions for digital certificates on the AS/400 system.

Of course, there are situations where standard management functions cannot be used or do not exist. The OS/400 uses application programming interfaces (APIs) to provide the needed functionality and flexibility for those applications. For example, if an HTTP server wants to authenticate a user checking the certificate in a validation list or if you want to manage user certificates in user applications, you have to use application programming interfaces (APIs). They can be used, for example, to:

- Check for user certificates in the validation lists
- Query user certificates based on their attributes
- Add certificates into validation lists
- Register applications for certificate use

Examples of how APIs may be used to build an application using certificates to provide security are provided in *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659.

Table 1 shows an overview of the available APIs with a brief description of each API. Note that APIs are available for the Original Programming Model (OPM) and Integrated Language Environment (ILE) environments.

Table 1. Digital certificate management APIs overview

API name	OPM name ILE name	Description
Add User Certificate	QSYADDUC QsyAddUserCertificate	Associates a certificate with an OS/400 user profile. Note that while a user profile may have many certificates associated with it, each certificate may be associated with only one user profile.
Add Validation List Certificate	QSYADDVC QsyAddVldlCertificate	Adds a certificate to a validation list. Validation lists may have many certificates and certificates may be added to many validation lists.
Check Validation List Certificate	QSYCHKVC QsyCheckVldlCertificate	Checks if a certificate is in a validation list.
Deregister Application for Certificate Use	N/A QsyDeregisterAppForCertUse	Removes an application and all associated certificate information from the registration facility.
Find Certificate User	QSYFNDCU QsyFindCertificateUser	Returns the user profile, if any, that is associated with a given certificate.
Find Validation List Entry	QSYFDVLE QsyFindValidationLstEntry	Given a validation list entry key, returns the validation list entry data and attributes.

API name	OPM name ILE name	Description
Find Validation List Entry Attributes	N/A QsyFindValidationLstEntryAttrs	Finds an entry in a validation list object and the attributes associated with the entry.
Find First Validation List Entry	N/A QsyFindFirstValidationLstEntry	Returns the validation list entry data for the first entry in the list.
Find Next Validation List Entry	N/A QsyFindNextValidationLstEntry	Given a validation list entry key, returns the validation list entry data for the next entry in the list.
List User Certificates (1)	QSYLSTUC QsyListUserCertificates	Creates a list of the certificates associated with a user profile and puts this into a user space.
List Validation List Certificates	QSYLSTVC QsyListVldlCertificates	Creates a list of the certificates in a validation list and puts this into a user space.
Open List of User Certificates	QSYOLUC N/A	Creates a list of the certificates associated with a user profile and returns this in a variable.
Parse Certificate (1)	QSYPARSC QsyParseCertificate	Parses a certificate and returns the internal fields in a variable.
Register Application for Certificate Use	QSYRGAP QsyRegisterAppForCertUse	Registers an application entry with the registration facility. This is a requirement to use SSL for application access.
Remove User Certificate	QSYRMVUC QsyRemoveUserCertificate	Removes the association of a certificate from a user profile.
Remove Validation List Certificate	QSYRMVVC QsyRemoveVldlCertificate	Removes a certificate from a validation list.
Note 1: New in V5R1, the QSYPARSC (QsyParseCertificate) and QSYLSTUC (QsyListUserCertificates) APIs have been modified to handle the e-mail attribute in user certificates. For details, refer to *Security APIs* found in the iSeries Information Center by clicking **Programming->CL and APIs->APIs by category->Security**.		

Refer to Chapter 3, “Object signing” on page 99, for a list of APIs that are available for object signing and digital signature verification.

For coding samples using the new Global Secure Toolkit (GSKit) APIs in sockets applications, refer to Chapter 6, “Using SSL in ILE RPG sockets applications” on page 335.

2.7 Backup considerations

Starting with V4R5, the way in which the certificate store’s passwords are held has changed. This requires a change to backup procedures. Now a single independent index per iSeries or AS/400 server is used to store encrypted key database passwords. Previously, the password was stored in IFS files that were accessible to any user with authority to the IFS file. Now it is not enough to save only the certificate store (.KDB and .RDB files) using the `SAV` command. To have a complete backup, you also have to save the independent index. The independent index is saved by using the `SAVSYS` and `SAVSECDTA` commands. It is restored by using the `RSTUSRPRF` command when USRPRF(*ALL) is specified.

Let’s see in more detail how the password is used and when it is retrieved from the index.

When you create a certificate store a password is required. The password is used to encrypt the data of the certificate store to protect the private keys.

When you want to manage the certificates, you have to open your certificate store by providing to the DCM the password to decrypt the key database files. Password checking does not involve the independent index. It is made directly on the certificate store by trying to decrypt the data with the password provided. If the password that you have given is correct, the DCM can decrypt the data and allow you to proceed. If not, an error message requires you to check the password and retry.

The independent index is involved when running SSL applications that need to access the certificate store and when resetting the password of a certificate store. The DCM has to access the password automatically in the independent index to open the certificate store, in the first case because the operation is unattended, and in the second one because the DCM tries to check the integrity of the certificate store itself.

Prior to V4R5, you had to make sure the matching IFS key database and stash file (.KDB, .RDB, and .STH) were saved together so they would match when restored. Now saves of the IFS key database need to remain

synchronously with the security data save. Anytime a key database password is changed, you should save the certificate store and the security data.

The correct way to move certificates is to export and import them. But if you choose to create a certificate store on a V5R1 or later system, and then copy that certificate store to another system for backup purposes, there will be no stashed password file to transfer to the target system. Or if for any reason the key database files and its password index entry are not correctly updated on your AS/400 system, you can generate the stashed password via DCM.

The stashed password can be generated by changing the password. You have to open the certificate store manually entering the password. As we mentioned earlier, password checking is done on the certificate store. Then in the Manage Certificate Store task, you must change the password. This step generates an entry in the password index for this certificate store and re-synchronizes the store with the index.

If you want to recreate the password on AS/400 systems with a release prior to V4R5, the stashed password IFS file can be created by selecting the option to change the certificate store password and specifying "yes" on the auto-login option.

To summarize, at V5R1, keep in mind these points:

- If you have forgotten the password of the certificate store, but you have a matching version of the password index, you can reset the password and then open the certificate store in the DCM.
- If you know the password of the certificate store, but you do not have a matching password index, you can set the entry in the password index by logging onto the certificate store and then selecting to change the password in DCM.
- If the password index does not match the certificate store and you have lost the password, then you have to recreate your certificate store, which means if you have not saved (exported) your certificates, they are lost.

The same password management was introduced in V4R5 by calling a program as described in 2.1.3.1, “Local Certificate Authority (CA)” on page 21.

2.8 Problem determination

In V5R1, the messages provided in the DCM tasks have been improved. Now the text gives you more information about the result of the operation

performed. At first sight, you can immediately identify a successful completion message because it will have a green background. Otherwise, the background is red. And in case of error messages, it also lets you know if additional messages are available in the ADMIN server job log or if a problem record was created.

For advanced DCM problem determination, starting with V4R4, there is a tracing facility for collecting documentation related to problems with certificate management processing. It is an advanced debugging tool, so it is expected to be required eventually by IBM Service Personnel for analysis. It is used for collecting service logs during the execution.

A trace and a dump file will be produced as a stream file and posted in IFS. The dump files are used to dump such error information as module name, exception line number, name with text, and primary and secondary return codes. The trace file is used as a flight recorder to log timestamps, type of trace record, thread ID, entries to routines, control blocks, and other miscellaneous trace information. When tracing for all certificate services, you also receive the trace spool files that are generated. The trace and dump files are named TRACEn.n and DUMPn.n, where *n* is a number. They have to be placed into the /QIBM/UserData/ICSS/CertSvcs/Log IFS directory.

If it is expressly required by the IBM Technical Support Center, the steps to collect the data will be:

1. Sign on to AS/400 with a user profile with *SERVICE special authority.
2. Delete or move any existing trace or dump files present in the /QIBM/UserData/ICSS/CertSvcs/Log IFS directory.
3. Start the tracing facility by entering the command:

   ```
   TRCTCPAPP APP(*CERTSRV) SET (*ON)
   ```

   You can also limit the trace to just DCM, SSL, object signing, and VPN key manager services by specifying the `CERTTYPE` parameter.
4. Recreate the problem.
5. Stop the trace by entering the command:

   ```
   TRCTCPAPP APP(*CERTSRV) SET (*OFF)
   ```
6. Collect the set of files created in the /QIBM/UserData/ICSS/CertSvcs/Log directory as well as the trace spool files. Give them to IBM. The IBM Service Representative will tell you what traces to collect.

2.8.1 SSL and sockets errors

For messages posted in QSYSOPR or any job logs that refer to SSL errors, you can enter the command:

```
DSPPFM FILE(QSYSINC/H) MBR(SSL)
```

It displays the SSL API definitions file that also contains the error codes and their meaning. Table 2 lists the return codes that were copied from an AS/400 system at V5R1. Similarly, for the error number, the command is:

```
DSPPFM FILE(QSYSINC/H) MBR(ERRNO)
```

Notice that the error numbers are not related just to the SSL environment, but to all the sockets environments.

Table 2. SSL return codes

Return code	Meaning
-1	SSL_ERROR_NO_CIPHERS
-2	SSL_ERROR_NO_CERTIFICATE
-4	SSL_ERROR_BAD_CERTIFICATE
-6	SSL_ERROR_UNSUPPORTED_CERTIFICATE_TYPE
-10	SSL_ERROR_IO
-11	SSL_ERROR_BAD_MESSAGE
-12	SSL_ERROR_BAD_MAC
-13	SSL_ERROR_UNSUPPORTED
-14	SSL_ERROR_BAD_CERT_SIG
-15	SSL_ERROR_BAD_CERT
-16	SSL_ERROR_BAD_PEER
-17	SSL_ERROR_PERMISSION_DENIED
-18	SSL_ERROR_SELF_SIGNED
-20	SSL_ERROR_BAD_MALLOC
-21	SSL_ERROR_BAD_STATE
-22	SSL_ERROR_SOCKET_CLOSED
-23	SSL_ERROR_NOT_TRUSTED_ROOT
-24	SSL_ERROR_CERT_EXPIRED

Return code	Meaning
-25	SSL_ERROR_BAD_DATE
-26	SSL_ERROR_BAD_KEY_LEN_FOR_EXPORT
-90	SSL_ERROR_NOT_KEYRING
-91	SSL_ERROR_KEYPASSWORD_EXPIRED
-92	SSL_ERROR_CERTIFICATE_REJECTED
-93	SSL_ERROR_SSL_NOT_AVAILABLE
-94	SSL_ERROR_NO_INIT
-95	SSL_ERROR_NO_KEYRING
-96	SSL_ERROR_NOT_ENABLED
-97	SSL_ERROR_BAD_CIPHER_SUITE
-98	SSL_ERROR_CLOSED
-99	SSL_ERROR_UNKNOWN
-1009	SSL_ERROR_NOT_REGISTERED
-1011	SSL_ERROR_NO_CERTIFICATE_AUTHORITIES

Chapter 3. Object signing

OS/400 already supports the use of certificates via Digital Certificate Manager (DCM) for various purposes. Until now though there has been no function to use them to protect the integrity of the code running on the iSeries of AS/400 server. This can be done by signing objects. Digital signatures are added to objects by a supplier so that the recipient can verify an object's integrity and its source.

At V5R1 the DCM interface was redesigned and enhanced in a number of ways. Several new advanced functions are now available including the ability to digitally sign objects. This chapter discusses the various steps necessary to set up and configure DCM to do object signing and signature verification.

DCM can now be used to create and manage certificates for digitally signing objects that will ensure their integrity and provide proof of origin. You can also create and manage the corresponding signature verification certificates used to authenticate and verify the signature on a signed object and ensure that the data in the object is unchanged.

3.1 Introduction

Beginning in V5R1, OS/400 provides support for using certificates to digitally sign objects and to verify the digital signature on them. As OS/400 objects are considerably more complex than a simple stream of bytes, normal signing tools do not work on them. Digitally signing an object provides a way to ensure the integrity of:

- The contents of that object
- The source of the object's origin

OS/400 objects have a different and more complex structure than simple stream files. An OS/400 object always consists of a header, a data part, and optionally other associated data spaces. Object signing as introduced in V5R1 signs only the parts of an object that are critical to the object's integrity. For example, if somebody changes the description text of an object, an integrity check would not show any violation.

An iSeries or AS/400 server user can use the GUI interface of DCM or the OS/400 command `CHKOBJITG` to verify the integrity or source of any part of an application if they have any concerns about it. Objects can also be checked when being installed on, or restored to the system. A user must have the correct level of authority to use a certificate to sign objects.

Objects stored on other systems or remote file systems cannot be signed by the source system. If a customer wants to do this they would have to set up object signing on the other system and sign from that one. In remote file systems, you might have to use other signing tools. Another way would be to sign on to the other system, save the objects to a save file, ship the save file to the local system, sign the objects, and then send them back using a save file.

Note

At V5R1, OS/400 has the ability to sign specific object types. It is worth noting the restriction, however, that at V5R1, OS/400 does not support signatures by *specific* user profiles. A signature on an object represents the system that signed it, not a specific user on that system who performed the signing operation.

You can use DCM to create and operate your own private CA and then use the CA to privately issue certificates for applications to use to digitally sign objects. Or you can use DCM to manage certificates that you purchase/obtain from a public Internet Certificate Authority (CA) for digitally signing objects. The process of signing certificates is the same regardless of whether you use public or private certificates.

A set of APIs that can be used to sign objects, verify their signatures, and retrieve characteristics of the object signatures is supplied at V5R1. You can work with object signatures manually, using the DCM, or you can write programs using these APIs to automate processes to do with object signing. The object-signing APIs are discussed later in this chapter.

To place a digital signature on an object, a certificate's private key is used to add an encrypted mathematical summary of the data in the object. The signature allows you to detect unauthorized changes to the object. The object and its contents are not encrypted or made private by the digital signature. However, the summary itself is encrypted to prevent unauthorized changes to it. To ensure that the object has not been changed in transit and that the object originated from an accepted, legitimate source, you can use the signing certificate's public key to verify the original digital signature. This process is called signature verification. If the signature no longer matches, the data may have been altered and you can request another copy of the signed object from the signer.

Certificates are stored in key database files (.kdb) in the Integrated File System (IFS) in the /QIBM/UserData/ICSS/Cert/Signing directory. There are two new .KDB (certificate databases) stream files:

- /QIBM/UserData/ICSS/Cert/Signing/SGNOBJ.KDB, which holds certificates and private keys for signing objects.
- /QIBM/UserData/ICSS/Cert/Signing/VRYOBJ.KDB, which holds certificates to verify object signatures.

Both these certificate databases are maintained using the DCM function of the system provided by option 34 of OS/400. For each of the certificate databases, there is also one database with the extension .RDB. These are certificate request databases used to store certificate requests, where the signed certificate has not yet been imported.

3.1.1 System state objects

Starting at V5R1, all operating system objects and licensed programs are signed when shipped. The necessary CA and signature verification certificate to verify OS/400 object signatures during installation time are also shipped with the OS/400 code. Thus, the signature verification for OS/400 system objects does not depend on the availability of DCM and other license programs that are usually required to perform signature verification. Therefore, even if DCM is not set up and there is no object signing or signature verification store IBM-supplied objects will be verified on restore.

3.1.2 Advantages of object signing

Digitally signing an object provides the following advantages:

- Ensures integrity of an object.
- Signing an object means that the recipient can be sure that no one has changed or manipulated the content of the material and that it is definitely from the claimed sender. This is achieved by digitally signing the object and then verifying it on the target system.
- Traditional controls cannot protect an object from unauthorized tampering while in transit across the Internet or other untrusted network or while the object is stored on a non-iSeries server. Using digital signatures on an object protects it from unauthorized changes.

3.1.3 Objects that can be signed

Objects that can be signed include:

- Save files (not empty ones) in the QSYS.LIB file system
- Programs of types *PGM, *SVRPGM, *SQLPKG, *JVAPGM and *MODULE, as well as stream files with attached Java programs
- IFS stream files in local file systems

Note that you cannot sign objects that are compiled for a release prior to V5R1.

All objects you want to sign must reside on a local file system. For example, if you operate a Windows 2000 server on the Integrated Netfinity Server (INS), you have the QNTC file system available in the IFS. The directories in this file system are not considered local, because they contain files owned by the Windows 2000 operating system. For example, if you could sign a program file in QNTC, the Windows 2000 server would not be able to process the program anymore since the program file would have a format that the PC operating system does not recognize.

To determine whether files are local you can use the Work with Object Links (`WRKLNK`) command as described in the following example.

1. Enter the command `WRKLNK OBJ('/QNTC/*')`

```
                              Work with Object Links

Directory  . . . . :   /QNTC

Type options, press Enter.
  2=Edit   3=Copy   4=Remove   5=Display   7=Rename   8=Display attributes
  11=Change current directory ...

Opt   Object link           Type     Attribute    Text
      AS1310SRVL            DDIR
      AS4ACONSOLE           DDIR
 8    BARLEN2               DDIR
      QRALYAS4A             DDIR
      QRALYAS4C             DDIR

Parameters or command
===> WRKLNK OBJ('/QNTC/*')
F3=Exit   F4=Prompt   F5=Refresh   F9=Retrieve   F12=Cancel   F17=Position to
F22=Display entire field           F23=More options
```

2. Select option `8` (Display attributes) for a displayed directory and press Enter.

```
                              Display Attributes

 Object . . . . . :   /QNTC/BARLEN2

 Text . . . . . . . . . . . . . . . . . :

                                                          Objects can be
 Type . . . . . . . . . . . . . . . . . :   DDIR          Local  - or - Remote

 Owner  . . . . . . . . . . . . . . . . :   QDFTOWN
 System object is on  . . . . . . . . . :   Remote
   Object overflowed  . . . . . . . . . :   No

 Coded character set ID . . . . . . . . :   0
 Hidden file  . . . . . . . . . . . . . :   No
 PC system file . . . . . . . . . . . . :   No
Readonly . . . . . . . . . . . . . : Yes
                                                                        More...
 Press Enter to continue.
 F3=Exit   F12=Cancel   F22=Display entire field
```

Only objects that are local and meet the other criteria described in this section are signable.

You can use the `WRKLNK` command with the options shown in the previous example for all files available in the IFS.

3.1.4 Signature removal

It is important to know that the use of certain commands will cause signatures to be removed from the affected object. The commands include:

- `CHGPGM` and `CHGSRVPGM` (except when changing the TEXT)
- `CHGMOD`, except when changing the TEXT and RMVOBS(*ILDTA)
- `UPDPGM, UPDSVRPGM`
- `RUNJVA` will remove the digital signature of a stream file if the command changes any of the Java programs.
- `CRTJVAPGM` and `CHGJVAPGM` will normally remove the digital signature of a stream file.

3.1.5 Certificate expiration date

When an object is signed, a certificate is used to store information about the signer. All certificates have finite lifetimes. When their expiration date is past, they can no longer be used when signing objects. The one exception to this is a verification certificate. When its expiration date is past, although it is outdated it will be tolerated and can still be used for verifying signatures.

3.1.6 Prerequisites

To configure object signing on an iSeries or AS/400 server, it must be at OS/400 Version 5 Release 1.

- The user profile of users who manage and set up object signing must have *ALLOBJ and *SECADM special authorities. This is required by interfaces that manage the registry of signers and verifiers.
- Users who use a particular signing application in DCM to sign objects can be authorized through the Operations Navigator Application Administration option.
- The new system value QVFYOBJRST must be set.
- Check QAUDCTL system value set with *OBJAUD. To get audit records cut for signing you need to have QAUDCTL set with *OBJAUD value and use CHGOBJAUD to specify object being signed.

To use object signing in OS/400, you need the following licensed programs:

- 5722-SS1 Option 34 Digital Certificate Manager
- 5722-TC1 TCP/IP Connectivity Utilities
- 5722-AC2 (56-bit) or 5722-AC3 (128-bit) IBM Cryptographic Access Provider.
- 5722-DG1 HTTP Server (for using DCM)

Since DCM is run through a Web browser interface, the HTTP ADMIN server must be running on the system.

3.1.7 Signing objects so that other operating systems can verify

Currently there is no global standard for signing objects, nor is there a global repository of certificates to make general exchange of signed objects possible. Object signing done using DCM or the appropriate API is unique to the iSeries and AS/400 server. If you want to sign objects in a way that other operating systems can verify object signatures, you will need to limit yourself to stream files (files not under /QSYS.LIB in the IFS file system). On stream files you can use such tools as the JARSIGNER utility that ships with Java 2 (on the iSeries or elsewhere) or use Signtool from Netscape.

Signtool can be downloaded from Netscape but runs only on AIX or Windows not on the iSeries itself. You can use it from a PC or UNIX system on file systems that have been mapped from the iSeries. This utility puts a parallel directory under the directory you are signing objects in that holds the object signatures.

3.2 Planning and usage considerations

The first step is to decide if the use of digital signatures will fit your security needs and policies. If it does, you will need to evaluate whether you should use public certificates as opposed to using certificates issued by private certificate authorities. OS/400 ships several certificates for well-known Certificate Authorities, for example VeriSign and Thawte.

If you are planning to distribute objects to users in the general public, you should consider using certificates from a well-known public Certificate Authority (CA), such as VeriSign. Certificates from well-known CAs are supplied with applications that will use them. When a public certificate is used, it ensures that others can easily and inexpensively verify the signatures that have been placed on objects that you distribute to them. If objects are going to be distributed solely within your organization, it may be worth considering using DCM to operate your own CA to issue certificates without purchasing them from an outside CA.

When digital signatures are verified you need to decide which Certificate Authorities you trust and which certificates will be trusted for signing objects. If you choose to trust a CA, you can also choose whether to trust signatures that were created by someone using a certificate that the trusted CA issued. If you decide not to trust a CA, this will imply that you do not trust certificates that the CA issues or signatures that a user created using those certificates.

3.2.1 Components of object signing and signature verification

There are various components and functions involved when implementing an object signing environment. Figure 56 on page 106 illustrates the various components involved when performing object signing and signature verification as far as the Digital Certificate Manager (DCM) is concerned.

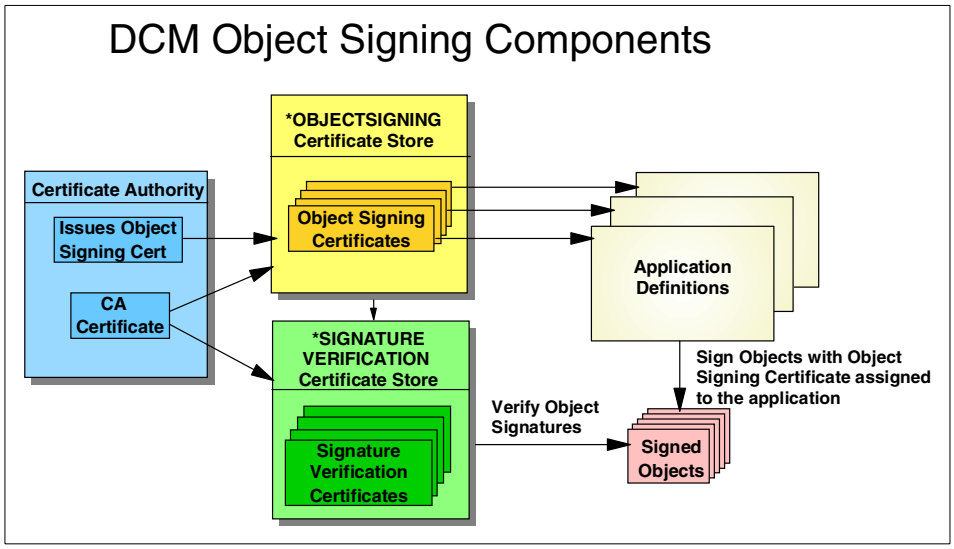


Figure 56. DCM object signing components

There are two certificate stores required on the iSeries server when performing object signing and signature verification tasks. The object signing certificate store (*OBJECTSIGNING) contains object signing certificates including the corresponding certificate's private key. It also contains the CA certificate of the CA that issued the object signing certificate. In order to sign objects, you also need to create an object signing application. This application does not represent an executable application, but rather defines an environment and description for the objects to be signed. You have to assign one object signing certificate to an object signing application to be able to sign objects.

The second certificate store, the signature verification store (*SIGNATUREVERIFICATION), holds signature verification certificates. These are basically object signing certificates without the corresponding private keys, because the private key is not required to verify object signatures. To be able to verify object signatures, you also need to have the corresponding CA certificate in the signature verification store. As shown in Figure 56, object signing is performed by using object signing certificates, while signature verification is performed by using signature verification certificates. In other words, even if the object signing certificate has the public key that is required to verify an object signature, you still need to set up the signature verification store and import the signature verification certificates to perform signature verification.

3.2.2 QVFYOBJRST system value and restore operations

When restoring objects onto your system you need to decide what level of checking is performed during restore operations by setting the new system value QVFYOBJRST.

QVFYOBJRST specifies the policy for object signature verification during restore or install operation. It controls how important signatures are for objects being restored onto your system. The default setting (3) allows unsigned objects to be restored, but ensures that signed objects can only be restored if the objects have a valid signature. The system defines an object as signed only if it has a signature that your system trusts. The system will ignore other untrusted signatures on the object and treat it as if it is unsigned. The following options are available:

1 Do not verify signatures on restore. Restore all objects regardless of their signature.

2 Verify signatures on restore. Restore unsigned user-state objects. Restore signed user-state objects, even if the signatures are not valid.

3 Verify signatures on restore. Restore unsigned user-state objects. Restore signed user-state objects only if the signatures are valid. This is the default.

4 Verify signatures on restore. Do not restore unsigned user-state objects. Restore signed user-state objects, even if the signatures are not valid.

5 Verify signatures on restore. Do not restore unsigned user-state objects. Restore signed user-state objects only if the signatures are valid.

Notes:

- When QVFYOBJRST is set to a value of 2 through 5, system state programs must always be signed when being restored.
- The values 4 and 5 requires that DCM is installed and set up on the restoring system.
- This system value has no effect on signed save files. Signatures on signed save files are always checked if they exist when attempting to restore an object from the save file. Signature failure on a save file will result in the restore operation failing. However, unsigned save files or save files where all signatures are untrusted are tolerated.
- QVFYOBJRST has no effect on most IFS files. Typically signatures on IFS files are not checked unless the `CHKOBJITG` command is run against it.

Some IFS files have attached *JVAPGM files and the system value controls the restoration of those IFS files. If this is the case, the IFS file is treated as a composite object, and if signature verification fails on one piece it results in the failure of the entire object.

- It is worth noting that if the system value has been set to disallow the restore of unsigned programs, if one *JVAPGM is unsigned, the IFS file and all attached *JVAPGMs will not be restored.

For more information about restoring objects, refer to 3.3.11, “Restoring objects with bad signatures” on page 172.

3.2.3 Check Object Integrity (CHKOBJITG) command

The `CHKOBJITG` command checks if a signature is valid for objects owned by a specific user profile or objects that match the specified path name for integrity violations. An integrity violation occurs when:

- The object has an incorrect domain attribute for its object type.
- An object (command, program or module) has been tampered with.
- An object has a digital signature that is not valid.

If an integrity violation occurs the object name, library name, object type, object owner and type of failure are logged to a database file - this is the outfile specified in the `CHKOBJITG` command. The type of violations that can occur are:

ALTERED The object has been tampered with

BADSIG The object has a digital signature that is not valid

DMN The domain is not correct for the object type

PGMMOD The runnable object has been tampered with

The database file also logs the following non-integrity violations as:

NOSIG Objects that do not have a digital signature but can be signed or objects that have a signature that cannot be verified due to an untrusted status. An untrusted status is when the signature verification store does not contain the object signer’s signature verification certificate.

NOTCHECKED Objects that could not be checked

Note

You must have *AUDIT special authority to check object integrity using the CHKOBJITG command.

The following parameters are displayed when checking for integrity violations using the CHKOBJITG command:

```
                    Check Object Integrity (CHKOBJITG)

Type choices, press Enter.

User profile, or . . . . . . . . USRPRF
Object . . . . . . . . . . . . . OBJ

File to receive output . . . . . OUTFILE
  Library  . . . . . . . . . . .                  *LIBL
Output member options:           OUTMBR
  Member to receive output . . .                 *FIRST
  Replace or add records . . . .                 *REPLACE
Check domain . . . . . . . . . . CHKDMN         *YES
Check program and module . . . . CHKPGMMOD      *YES
Check command  . . . . . . . . . CHKCMD         *YES
Check signature  . . . . . . . . CHKSIG       > *SIGNED
```

Figure 57. Parameters of CHKOBJITG command

You must specify a value for either the USRPRF or OBJ parameter as well as the outfile that will receive the verification results.

You will need to decide which options you want for the CHKSIG parameter. If you select *SIGNED only objects with digital signatures are checked. Any object with a signature that is not valid will be logged. If you select *ALL, all objects that can be digitally signed are checked. Any object that can be signed but has no signature, as well as any object with a signature that is not valid, will be logged.

When the CHKOBJITG command completes, you will normally see one of the following messages at the bottom of the window:

CPC2229 `Command completed successfully but violations found`. If you get this message, one or more violations were found and entries added to the results file specified.

CPC2230 `Command completed successfully`. This means that the integrity check resulted in no violations and no result file has been created.

Note

No successes are logged using the CHKOBJITG command, but only integrity violations are shown, for example, if an object does not have a valid signature.

To look at the results file (OUTFILE parameter called TESTOBJ) specified for receiving output, you can use the following command:

```
DSPPFM FILE(VANESSA/TESTOBJ)
```

On the right of the window, you will see the signed status of particular objects:

```
 Display Physical File Member
File . . . . . . :   TESTOBJ            Library  . . . . :   VANESSA
Member . . . . . :   TESTOBJ            Record . . . . . :   1
Control  . . . . .                      Column . . . . . :   1
Find . . . . . . .
*...+....1....+....2....+....3....+....4....+....5....+....6....+....
1030501104936AS4A    1                   *STMF   VANESSA   NOSIG
1030501104936AS4A    1                   *STMF   VANESSA   NOSIG
1030501104936AS4A    1                   *STMF   VANESSA   NOSIG
1030501104936AS4A    1                   *STMF   VANESSA   NOSIG
1030501104936AS4A    1                   *STMF   VANESSA   NOSIG
1030501104937AS4A    1                   *STMF   VANESSA   NOSIG
1030501104937AS4A    1                   *STMF   VANESSA   NOSIG
1030501104937AS4A    1                   *STMF   VANESSA   NOSIG
1030501104937AS4A    1                   *STMF   VANESSA   NOSIG
1030501104937AS4A    1                   *STMF   VANESSA   NOSIG

F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys
```

Figure 58. Results displayed from the CHKOBJITG command

You can use the F20 key to display the related object names.

Note that running the CHKOBJITG command with CHKSIG(*ALL) specified without having a signature verification store properly set up, the result file contains an entry with a type of NOSIG for each of the checked objects.

3.2.4 Display Object Description (DSPOBJD) command

You can use the DSPOBJD command to see if an object in the QSYS.LIB file system is digitally signed. However this CL command does not check the validity of the signature.

Enter the following command and specify the object to display:

```
DSPOBJD OBJ(AS4AUSER/TEST2) OBJTYPE(*FILE)
```

Enter 5 to display the full attributes. Page down to see if the object is digitally signed.

```
Display Object Description - Full

Object . . . . . . . :   TEST2          Attribute  . . . . . :
  Library  . . . . . :     VANESSA      Owner  . . . . . . . :
Type . . . . . . . . :   *FILE          Primary group  . . . :

Change/Usage information:
  Change date/time . . . . . . . . . :   03/05/01  15:49:14
  Usage data collected . . . . . . . :   YES
  Last used date . . . . . . . . . . :   02/15/01
  Days used count  . . . . . . . . . :   2
    Reset date . . . . . . . . . . . :
  Allow change by program  . . . . . :   YES
Auditing/Integrity information:
  Object auditing value  . . . . . . :   *USRPRF
  Digitally signed . . . . . . . . . :   YES
```

Figure 59. Window displayed after issuing the DSPOBJD command

3.2.5 Work with Object Links (WRKLNK) command

The `WRKLNK` command examines the attributes of an object in the IFS and indicates if it is digitally signed. Let's say we wanted to check if a particular object in the /payroll/InsData/setup.inf directory was digitally signed.

1. Type in the following:

   ```
   WRKLNK OBJ('/payroll/InsData/setup.inf')
   ```

2. Enter option 8 to display the object attributes.
3. Page down to see if it is digitally signed.

```
                                Display Attributes

Object . . . . . :   /payroll/InsData/setup.inf

Last used date . . . . . . . . . . . . :   03/26/01
Days used count  . . . . . . . . . . . :   1
  Reset date . . . . . . . . . . . . . :

Allow write during save  . . . . . . . :   No

Digitally signed . . . . . . . . . . . :   Yes

File ID  . . . . . . . . . . . . . . . :   X'0000000000000000181EAC85B000069D5'
```

Figure 60. Using WRKLNK to check if object is digitally signed

3.2.6 How objects are shipped/transferred

There are various ways of shipping objects from the signing system to a destination system. You have to be careful about the method you choose to transfer an object to ensure that object signatures are not removed.

3.2.6.1 Methods to retain the object signature

The only way to keep the signature on an object when shipping or transferring it to another system is by saving it to a save file or media, such as a tape. When saving the object in a save file, you can either FTP the save file to another system or use the Send Net File (SNDNETF) command.

3.2.6.2 Methods that remove the object signature

FTPing stream files or using the copy-and-paste function on mapped drives to transfer objects to another system will remove the object's signature. The reason for this is that an OS/400 object signature could cause problems on a target system if this system does not support OS/400 object signature.

3.3 Using object signing and verification

To digitally sign objects and verify their signatures, you use DCM to set up and configure the relevant certificate stores and certificates. These can either be exported to other systems and used for object verification or used to verify signature on objects on the source system.

In this section let us assume you are a Business Partner who wants to use object signing on your application objects. This is necessary as they are being sent to customers across an untrusted network. Your customers have iSeries and AS/400 systems as well. You have an application to send the

objects to remote customers automatically and want to digitally sign them before they are sent. Figure 61 shows the implementation flow of object signing and signature verification as described in this chapter.

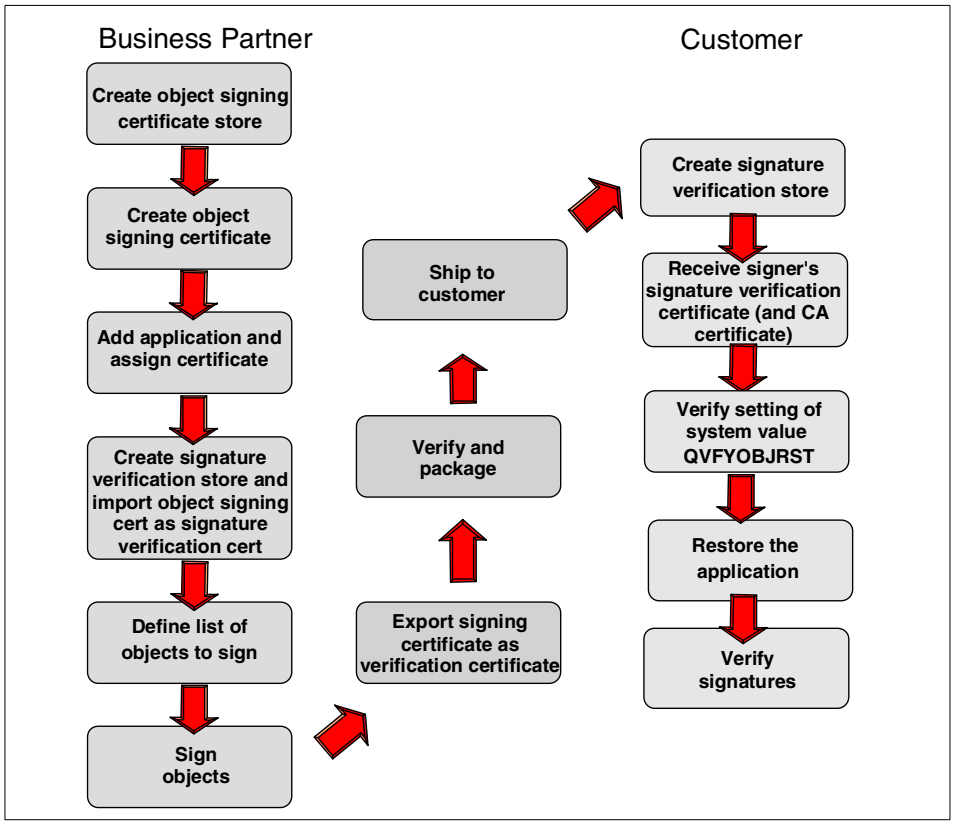


Figure 61. Implementation flow

3.3.1 Scenario objectives

Digitally signing an object allows the recipient to verify an object's integrity and its source. It allows the target system to determine unauthorized tampering with objects while in transit across the Internet or other untrusted network. A software vendor, for example, will use object signing on its own system and signature verification on its customers' systems.

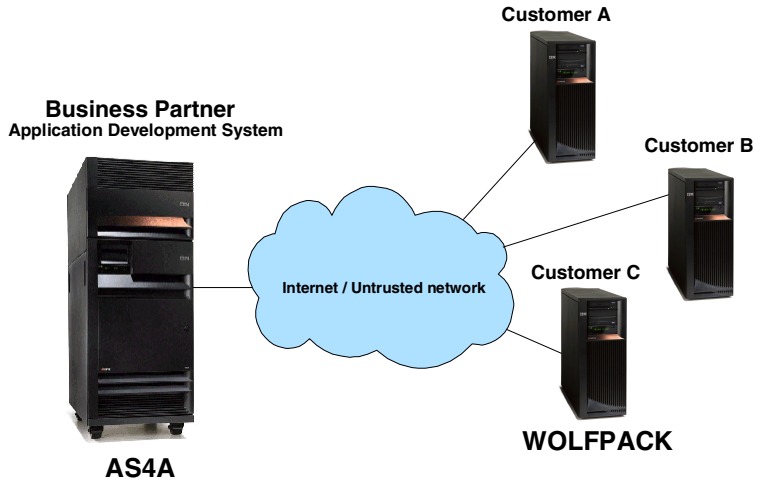
Use DCM to set up object signing on the system. As a Business Partner or vendor you will need to follow the steps and tasks in the next few sections to ensure object signing completes successfully on your side and signature

verification is run and completes without any integrity violations on the remote or receiver's system.

The objective for you is to provide your customer with proof of origination and allow recipients or the receiving system to verify the objects to ensure that they have not been changed while in transit.

3.3.2 Scenario environment

The following section talks about the tasks you, as a Business Partner called Smith and Johnes Ltd., would need to perform for setting up DCM on the local or source system to use it for digitally signing objects. The sections also cover the setup on the customer system to verify the object's integrity and origin.



	Payroll application		Banking application		Sales application	
Objects						
	Lib PAYSLIP	All objects	Lib BNKOBJ	All objects	Lib SLSOBJ	All objects
	Dir /payins1	insopt.cfg	Lib BNKDTA	All objects	Lib SLSDTA	All objects
	Dir /payfiles	All *.INS	Dir /bnkins	insopt.cfg	Dir /slsins	insopt.cfg
Application administrator						
	User	MARION	User	CLARISSA	User	RICARDA

Figure 62. Object signing Business Partner scenario

As shown in Figure 62 the Business Partner has an iSeries server AS4A that is used by programmers to develop payroll, banking, and sales applications. You can also see for each application the objects that make up the application package. There is one user per application that is responsible for administering the application packaging and shipping. This user will also sign the objects for the application the user is in charge for.

In this scenario we will show you how to set up object signing and signature verification for the payroll application. It covers the configuration for the development system AS4A and one customer system WOLFPACK.

3.3.2.1 Setting up the object signing environment

Before using DCM (or Sign Object API) to sign objects, you must ensure that certain prerequisite conditions are met. You will need to follow the tasks below to ensure object signing completes successfully.

1. Your iSeries or AS/400 server and those of your customers must be at least at OS/400 Version 5 Release 1.
2. You must have created the *OBJECTSIGNING certificate store, either as part of the process of creating a private CA or as part of the process of managing object signing certificates from a public Internet CA.
3. The *OBJECTSIGNING certificate store must contain at least one certificate, either one that was created by using a private CA or one obtained from a public Internet CA.
4. At least one object signing application must have been created to use for signing objects.
5. You must have assigned a specific certificate to the object signing application definition that you plan to use to sign objects.
6. The system used to sign certificates must also have an associated private key to do the signing operation. The private key is usually created when creating the certificate request.
7. The system verifying the resulting signature needs a copy of the public key to decrypt the work of the private key.
8. The system must be set up so that only the person who is in charge of signing and packaging the application can sign the application objects for an individual application.

3.3.2.2 Signing and packaging

The following tasks need to be performed by the application administrator to sign objects and package the application:

1. Sign all signable application objects.
2. Package the application.
3. Ship the application.

3.3.2.3 Setting up the signature verification environment

The following prerequisites have to be met on the customer systems to be able to verify the signatures on the Business Partner's application objects.

1. The iSeries or AS/400 server must be at least at OS/400 Version 5 Release 1.
2. You must have created the *SIGNATUREVERIFICATION certificate store.
3. The *SIGNATUREVERIFICATION certificate store must contain the CA certificate that signed the object signing certificate. Many well-known CA certificates are shipped with OS/400. Private CA certificates have to be imported. The *SIGNATUREVERIFICATION certificate store must also contain the signature verification certificate that was created and exported on the signing system.
4. The QVFYOBJRST system value must be set correctly.

3.3.2.4 Restoring the application

A user profile that restores the application needs *AUDIT special authority to perform the signature verification.

3.3.3 Setting up the development system for object signing

This section describes the steps necessary to set up DCM if you as the Business Partner want to use a public CA for object signing. The Payroll application consists of:

- A library PAYSLIP with many objects where you need to sign all signable objects.
- An IFS directory payins1 with a single object insopt.cfg that needs to be signed.
- An IFS directory payfiles where all files with the extension .INS need to be signed.

3.3.3.1 Managing public Internet certificates for signing objects

Your company Smith and Johnes Ltd. has customers situated all over the world. In this case it would be impractical for you to use your own private CA to issue certificates for object signing and distribute the CA certificates to all your customers to be able to verify the signatures on your objects sent to them. You have therefore decided to use a public CA for object signing.

Since you have decided to use DCM to manage public Internet certificates for your applications to use for object signing, but do not use DCM to operate your own CA, you will have to create the appropriate certificate store for managing the public certificates that will be used for object signing. This is the *OBJECTSIGNING certificate store. DCM takes you through the process of creating:

- The *OBJECTSIGNING certificate store.
- The certificate request information that has to be provided to the public CA to obtain a certificate

To use a certificate to sign objects, you will also need to define an application ID. This controls the level of authority required for someone to sign objects with a specific certificate.

Using DCM to manage a public Internet certificate to sign objects
The following tasks must be completed:

1. Use a Web browser and enter the following URL to display the AS/400 Tasks page:

   `http://servername:2001)`

   This requires that the HTTP ADMIN server instance is up and running. When prompted, sign on with a user profile that has *ALLOBJ and *SECADM special authorities. Note that this authority level is only required to set up the object signing environment. For signing objects you can use a user profile without any special authorities. You just need to authorize the user profile to the signing application using Operations Navigator.
2. Click **Digital Certificate Manager** from the AS/400 Tasks page.
3. Click **Create New Certificate Store** in the left-hand navigation pane.

   Note

   If you have any questions about how to complete a specific form in this guided task, select the question mark (?) in the top left of the window.

4. Choose ***OBJECTSIGNING** as the certificate store to create and click **Continue**. If the *OBJECTSIGNING option is not shown, the *OBJECTSIGNING store already exists.

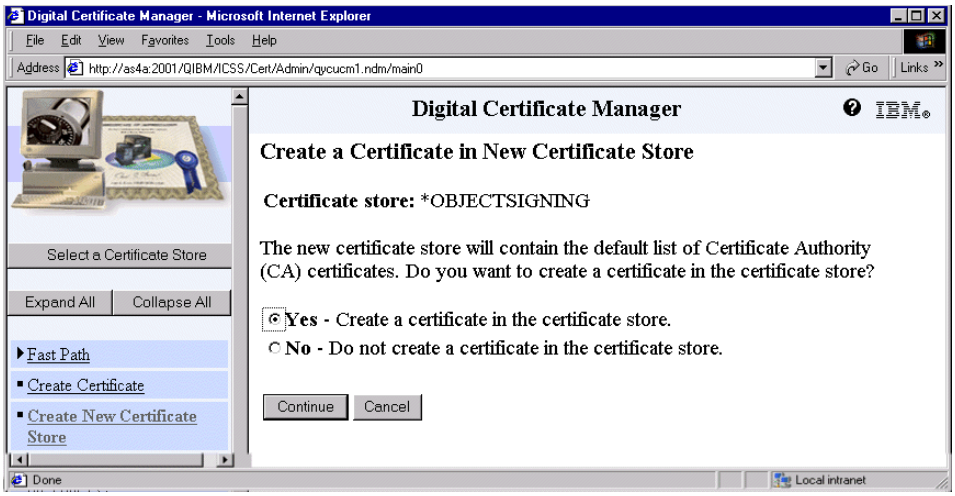


Figure 63. Create a Certificate in New Certificate Store window

5. Select **Yes** to create a certificate as part of creating the certificate store and click **Continue.**

   If you select No, an empty certificate store will be created and you have to create the certificate request afterwards or import an object signing certificate at a later time.

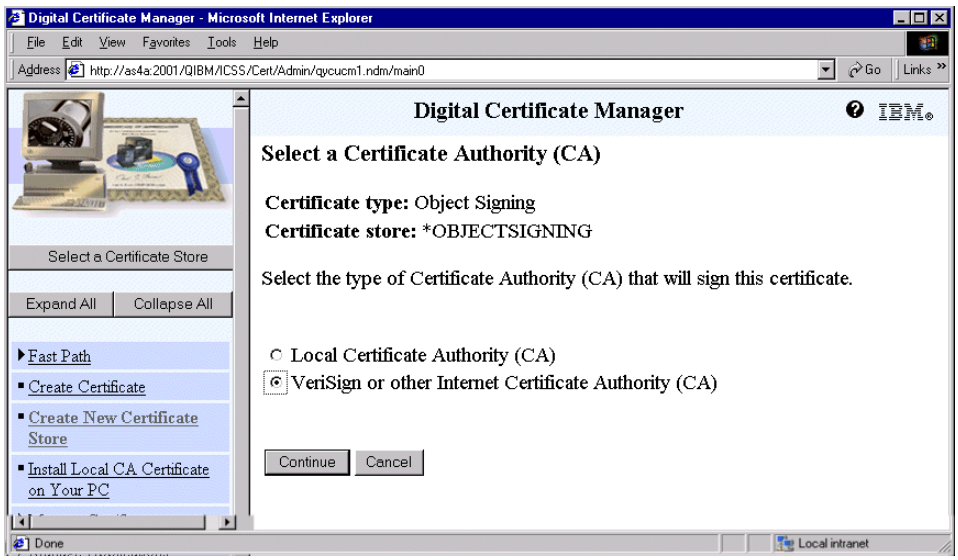


Figure 64. Select a Certificate Authority window

Select **Local Certificate Authority (CA)** if you want your local CA to sign the certificate request. This option is not available if you do not have set up a local CA on this system. Appendix D, “Creating a local Certificate Authority” on page 429 shows the steps for setting up a local CA including the steps for creating the system and object signing certificate store.

Select **VeriSign or other Internet Certificate Authority (CA)** if you want a public Internet CA to sign the certificate request. DCM generates information that you can use to request a certificate from VeriSign or another public CA. This information is also referred to as a Certificate Signing Request (CSR).

6. Select **VeriSign or other Internet Certificate Authority (CA)** as the signer of the new certificate and click **Continue**.
7. A form is displayed as shown in Figure 65.

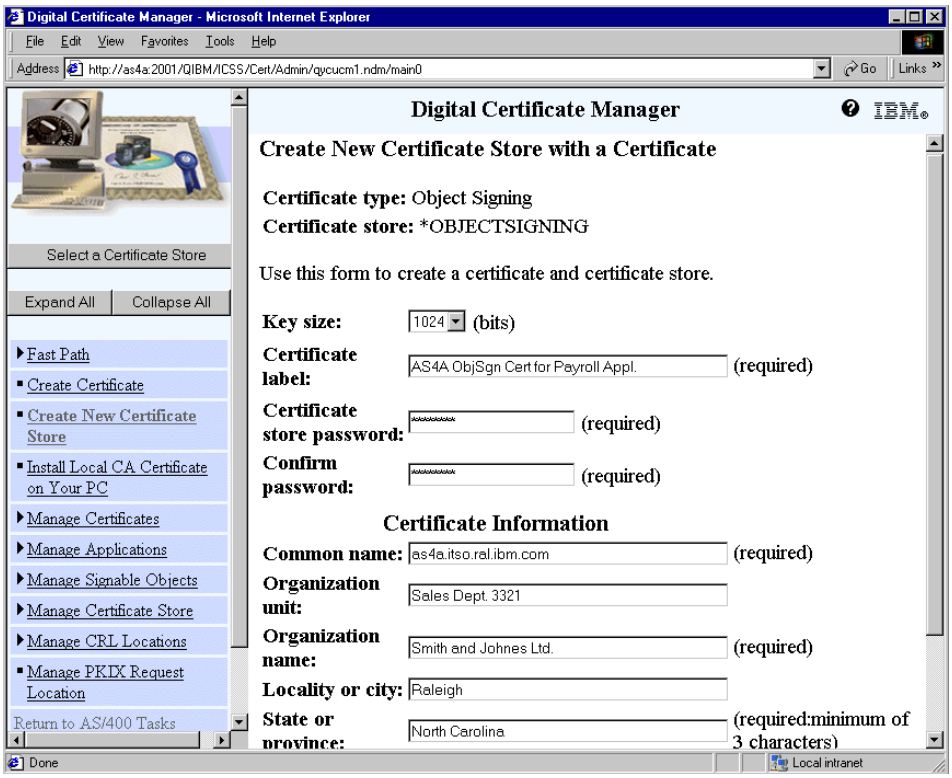


Figure 65. Create New Certificate Store with a Certificate

Complete the following fields to provide identifying information for the new certificate:

Certificate label Enter a name for the new certificate. This name uniquely identifies the certificate in the certificate store. You must specify a unique label.

Certificate store password This is a required field. Passwords can be from 1 to 128 characters, are case sensitive and can contain only alphanumeric characters. This password is used to protect the object signing certificate store. Without it you will not be able to access the object signing store.

Confirm password This is a required field. Just re-enter the password to validate it.

Common name Enter a name to describe the certificate application. This is a required field.

Organization unit This is an optional field.

Organization name This is a required field.

Locality or city Enter a name for the location this system is located in.

State or province Enter your state or province. It must be a minimum of three characters. Note that if you do not type in the full name, it may not be recognized by the public CA.

Note

Attempts to request a certificate from VeriSign failed several times because an abbreviated version was entered for the State or province field in the Create New Certificate Store with a Certificate window, which VeriSign did not recognize. Ensure the state or province name is typed in full.

8. Click **Continue** on the bottom of the window. This displays the certificate signing request (CSR) data that the public CA will need in order to issue your certificate. The CSR consists of the public key and other information that you have specified as the subject information for the new certificate.

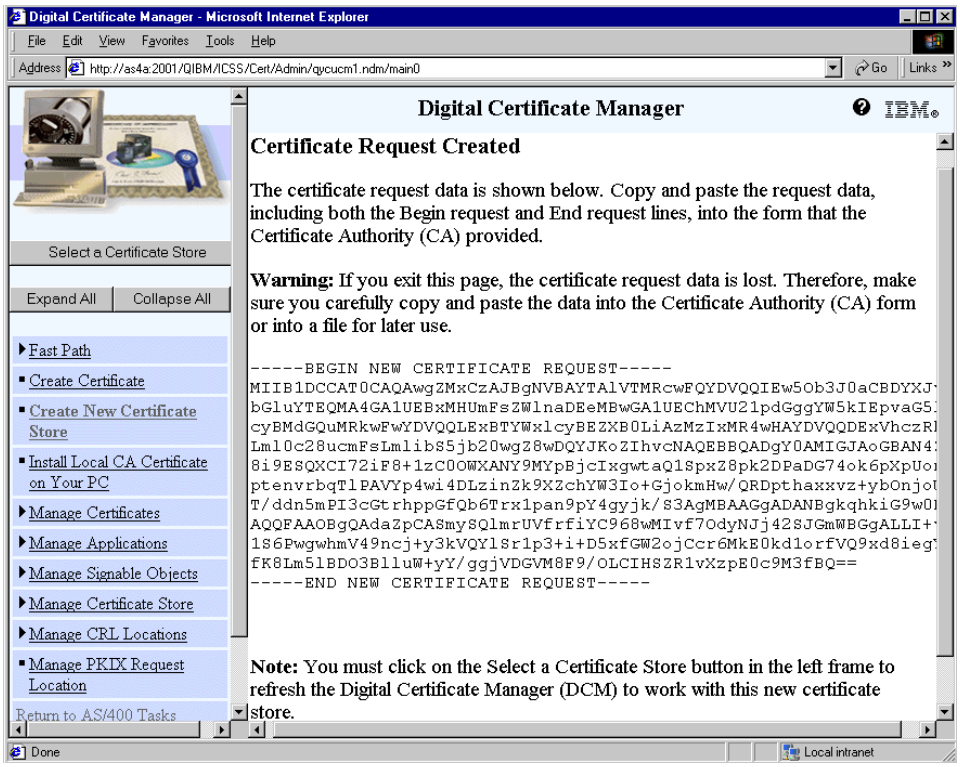


Figure 66. Certificate Signing Request data required by public CA

9. Go to the Web site of the public CA that you are going to use and follow the CA's instructions on how to provide the CSR data. Usually this is done by copying and pasting the CSR data that the public CA requires for requesting a certificate into the certificate application form or into a separate file. Be careful to copy all the CSR data including the Begin and End New Certificate Request lines. Send the application form or file to the CA that you have chosen and wait for the CA to return the signed and completed certificate before progressing with this procedure.
10. Click **OK** to close the certificate request window.

After receiving the signed certificate from the public CA

After receiving the signed certificate from VeriSign, for example, you need to copy the signed certificate into a text file on the iSeries server in a directory in the IFS. Perform the following steps to receive and import the signed certificate.

1. Open Windows Explorer.
2. Create a text file in your directory for example /verisign.txt or as we did a file named /barlen/PayrSgn.txt. Copy the signed certificate from the public CA and paste into the text file you have created.
3. Start DCM from a Web browser session by entering the following:
   `http://servername:2001`

   When prompted sign on to the AS/400 Tasks page.
4. Click **Digital Certificate Manager**.
5. From the left-hand navigation pane, click **Select a Certificate Store** and select ***OBJECTSIGNING** as the certificate store to open.

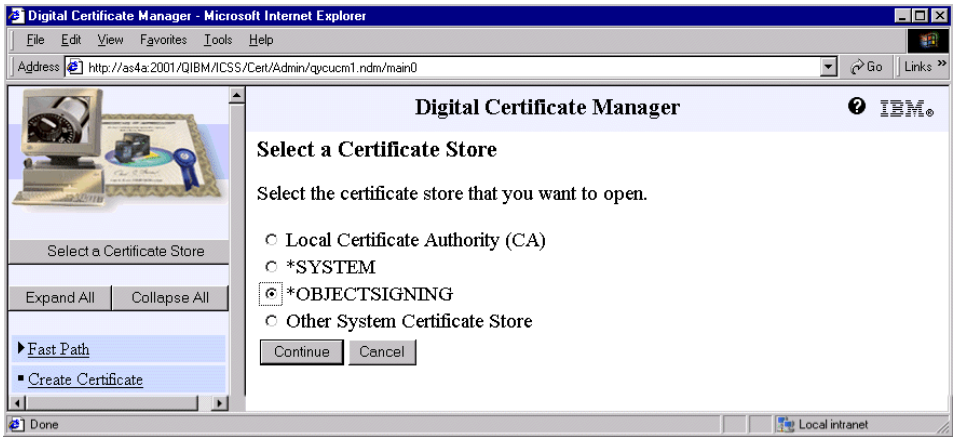


Figure 67. Select a Certificate Store window

6. Click **Continue**.
7. Enter the password that you specified for the certificate store when you created it and click **Continue**.
8. From the left-hand pane of the DCM window, select **Manage Certificates** to display a list of tasks shown in Figure 68.

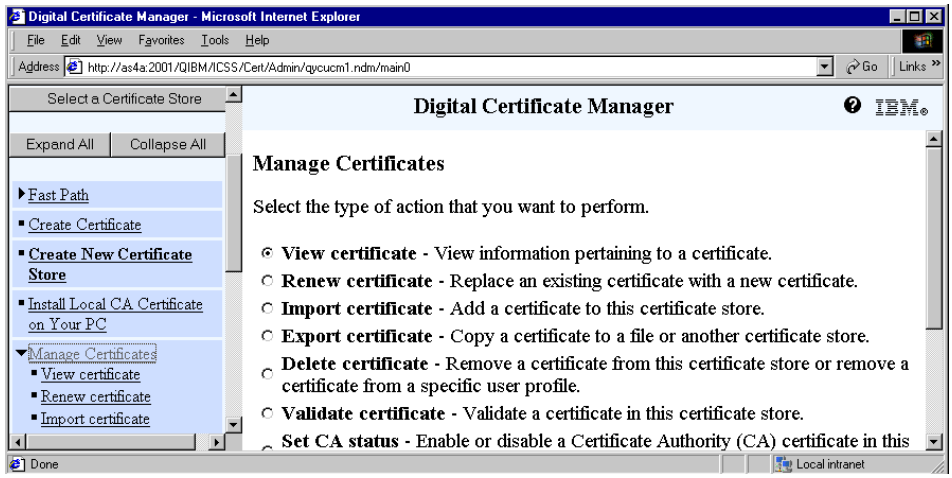


Figure 68. Manage Certificates window

Manage certificates

The following options are available:

View certificate Allows you to look at the distinguished name and other information for a certificate in the current store. The type of certificate that you can view varies based on the certificate store you are working with.

Renew certificate Allows you to renew an existing, expiring certificate. The type of certificate that can be renewed varies and depends on the certificate store that you are working with. This option is not available for the *SIGNATUREVERIFICATION certificate store.

Import certificate Is used to add a certificate to the current certificate store. The type of certificate that can be imported varies and is based on the certificate store that is being worked with. This option is used to import entire certificates including private keys that were exported in PKCS12 format on another system or for importing a signed certificate request.

Export certificate Copies a certificate from the current certificate store to a file or another certificate store. Again the type of certificate that can be exported varies based on the certificate store that you are working with.

Delete certificate Allows you to delete a certificate from the current certificate store, or to remove a certificate from a specific user profile.

Validate certificate Validates the authenticity of a certificate in the current certificate store. The type of certificate that you can validate varies based on the certificate store with which you are working. By default, only the expiration date and the certificate signature is validated. You can also change the configuration so that a Certificate Revocation List will be contacted to check whether the certificate has been revoked.

9. From the task list select **Import certificate** and click **Continue** to display the following window.

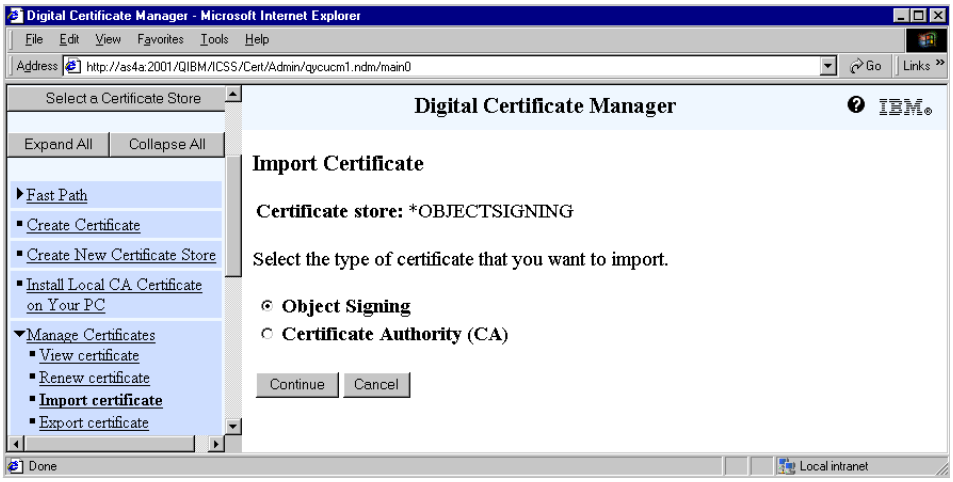


Figure 69. Import Certificate window

10. Select **Object signing** as the type of certificate you want to import and click **Continue**.

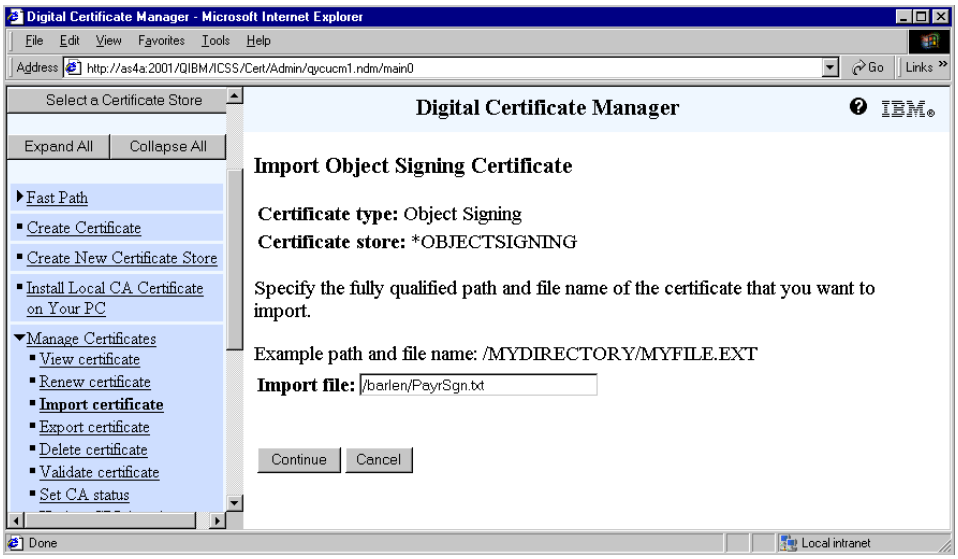


Figure 70. Import Object Signing Certificate window

11. In the window shown in Figure 70, enter the fully qualified path and file name of the certificate that you want to import. In this scenario we entered `/barlen/PayrSgn.txt`.

12. Click **Continue** to complete the import.

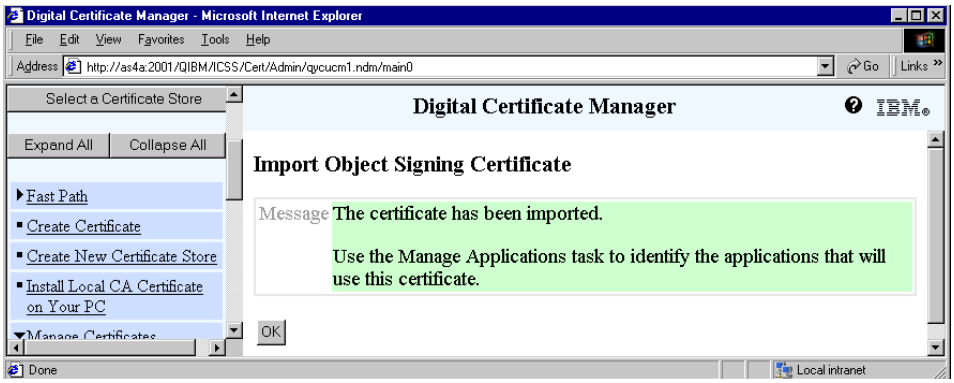


Figure 71. Import Object Signing Certificate confirmation window

A confirmation message (shown in Figure 71) is displayed. Note a green background confirms a successful completion.

When this has finished, you need to create an application definition for using the certificate to sign objects.

13. Select **Manage Applications** to display a list of tasks, as shown in Figure 72.

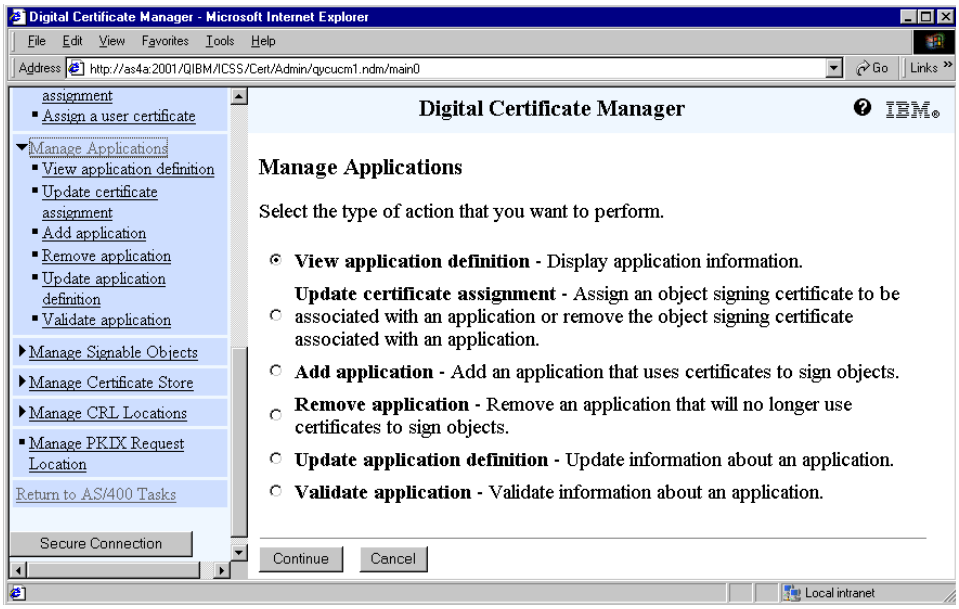


Figure 72. Manage Applications window

The tasks are:

View application definition
Select this to view information for an application.

Update certificate assignment
Select this to change the certificate assignment for an application.

Add application Select this to add an application definition to the DCM application database.

Remove application
Select this to remove an application definition from the DCM application database.

Update application definition
Select this to change the definition information for an existing application.

Validate application
Select this to validate information for an application

definition. This also validates the certificate assigned to the application.

14. From the task list in the right pane, select **Add application** to create an object signing application definition that will use a certificate to sign objects. The application definition and ID are used to describe the type of objects that you are planning to sign with a specific certificate. The online help (?) will assist you in completing the form.

    Do not get confused with the term *application*. The application as defined in DCM has no direct relationship to a software application package, such as the payroll application. It is used to define a signing environment, which specifies, for example, what certificate is used to sign objects or whether CRL checking is supported.

15. Click **Continue** to display the form in Figure 73.

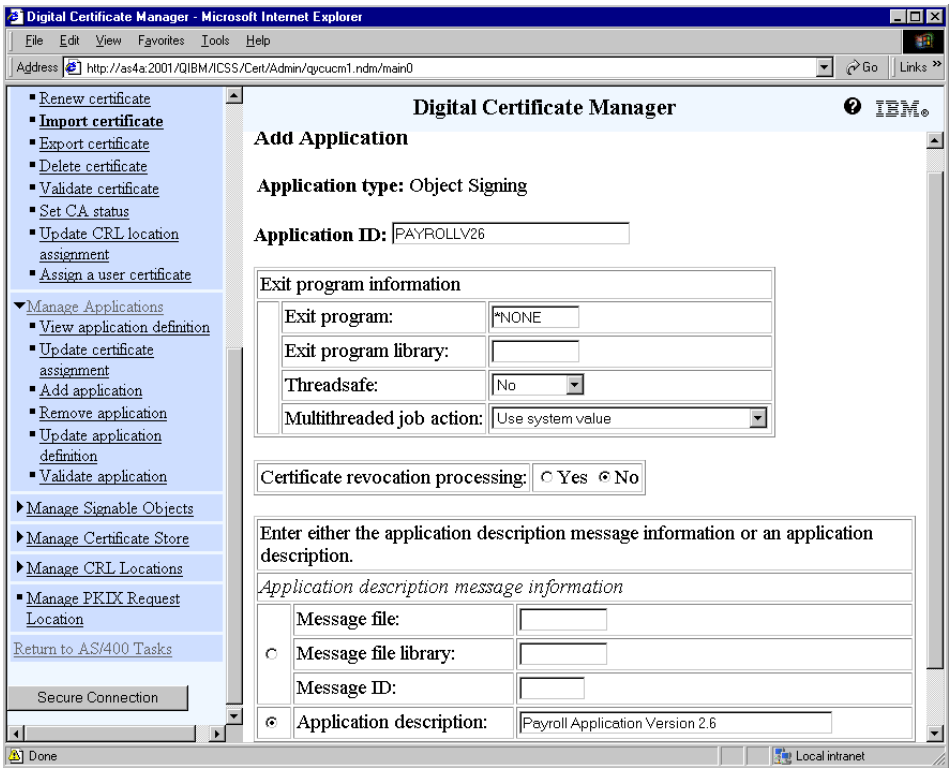


Figure 73. Add Application window

Application ID: The application identifier is used by DCM to associate the application with an object signing certificate. Object

Signing APIs also use the application ID when accessing the object signing certificate store to retrieve the object signing certificate. It is required and must consist of letters, numbers, underscores, or periods. The first character must be a letter. All lowercase letters will be changed to uppercase.

Exit program information:
You probably want to specify an exit program. The program named here can be used to protect your application. It is called when requests to change your application definition in the DCM, or to change the certificate assignment, are performed. The program can then notify, for example, a security administrator that somebody changed the application definitions. The exit program is also called when an attempt is made to de-register the application. In this case the exit program can refuse or accept the de-register request. The exit program is not called as part of certificate processing.

Certificate revocation processing:
Set this to **Yes** if you want to perform CRL checking on certificates used to sign objects. For more information on how to set up CRL processing on the iSeries server, refer to Chapter 2, “Digital Certificate Manager” on page 11.

Application description
This describes the object signing environment and what kinds of objects you want to sign with this application. The application description can be provided as a reference to the description by entering the message ID and message file, or by entering an application description text.

16. Complete the form and click **Add**. You should get the confirmation window shown in Figure 74.

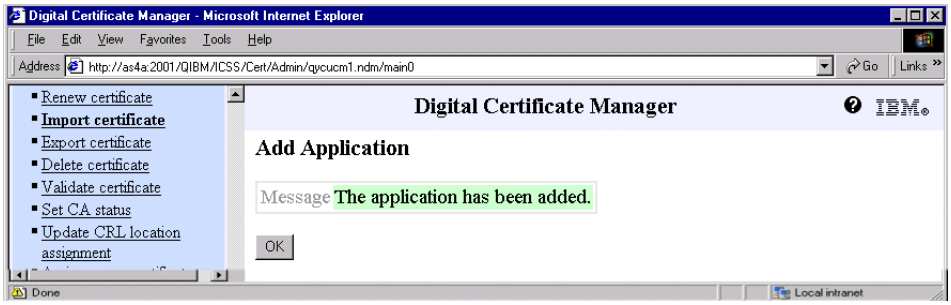


Figure 74. Confirmation window when application has been added

17. Click **OK**.

18. From the task list in the right pane of the window, select **Update certificate assignment** and click **Continue**. This will display a list of object signing application IDs to which you will assign a specific certificate.

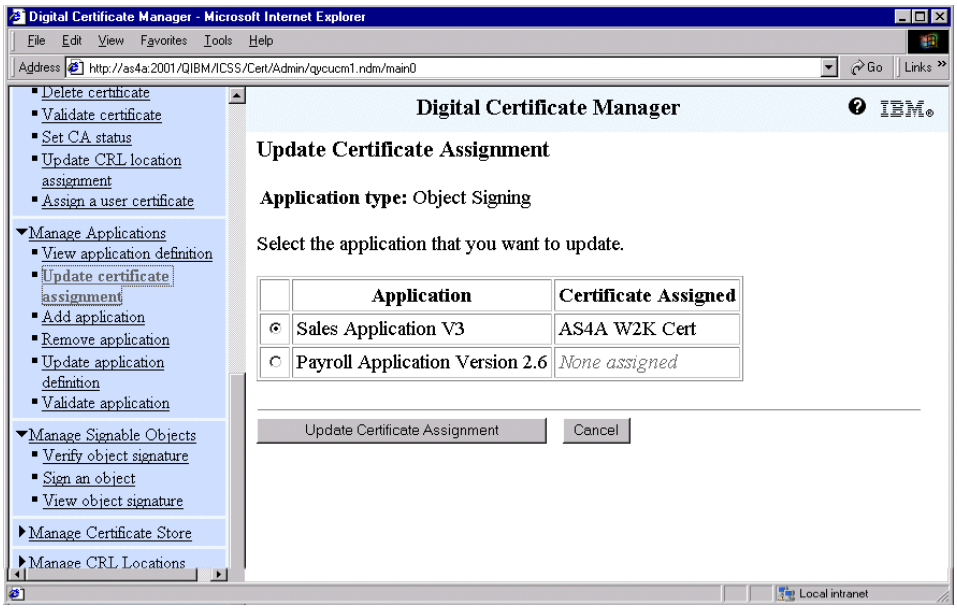


Figure 75. Update Certificate Assignment window

Update Certificate Assignment

Select this to add, change, or remove the current certificate assignment for the selected application.

Note

You may have noticed in this scenario that even when we created a new object signing certificate store and defined only one application, there are two applications shown. This is because the application definitions are not stored in the object signing certificate store. They are stored in a different place. The sales application shown in Figure 75 was defined during previous tests. If you want to delete an application definition you have to select the **Remove application** option from the Manage Application tasks.

19. Select your application ID and click **Update Certificate Assignment**.

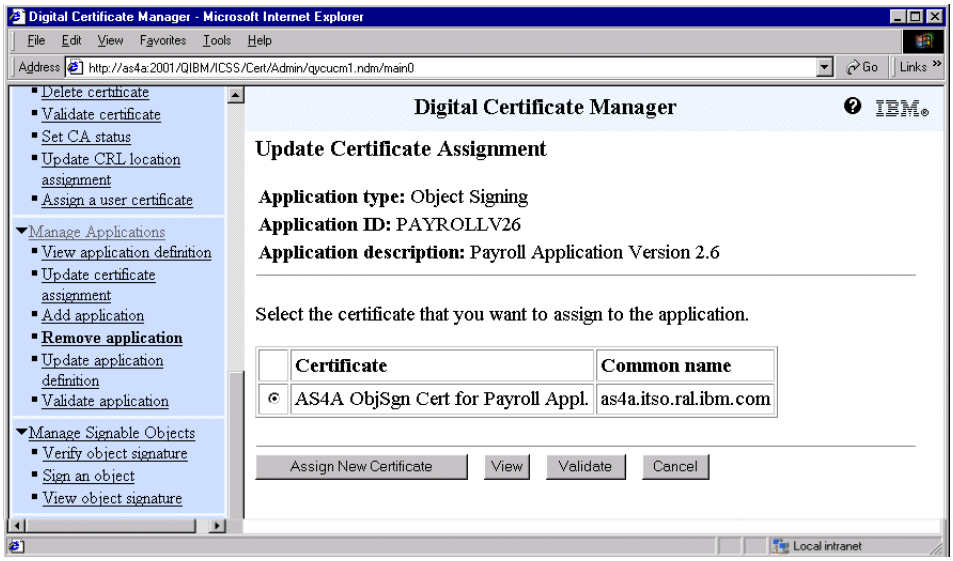


Figure 76. Assign New Certificate

20. Select the certificate that you want to assign to the application. In this case it is the certificate that was requested from VeriSign. Click **Assign New Certificate**.

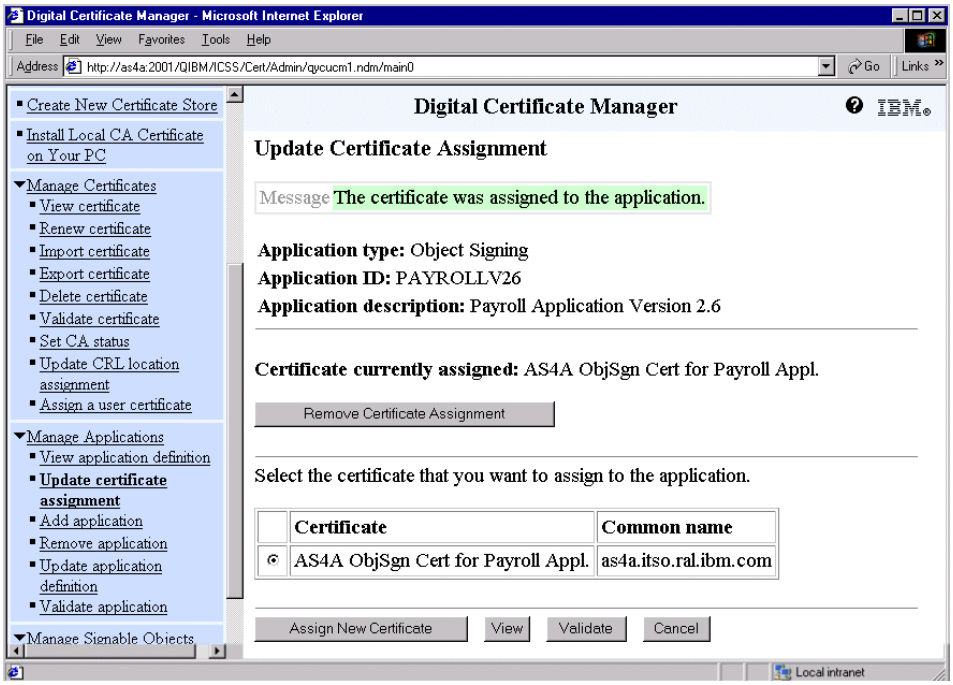


Figure 77. Update Certificate Assignment confirmation window

The application has a certificate assigned and is ready to sign objects. However, the setup process is not completed yet. As a part of the setup you also have to export the object signing certificate so that others can later on, for example during restore, verify the object signatures.

3.3.4 Exporting object signing certificates for verification

For the application administrator on the development system to verify object signatures before packaging the application, a signature verification store needs to be created. Then the object signing certificate must be exported from the object signing store to the new signature verification store as a signature verification certificate. The signature verification certificate must also be made available to be sent to the customer system. Without having the signature verification on the customer system, the application object signatures cannot be verified.

Perform the following steps to create the signature verification environment on the development system and to make the object signing certificate available as a signature verification certificate to be used on other systems.

1. Click **Create New Certificate Store** from the DCM navigation pane.

   Note that creating the signature verification store is not required to sign objects. However, we recommend that you also set it up on the development system to verify the signatures a last time before packaging and shipping the application.

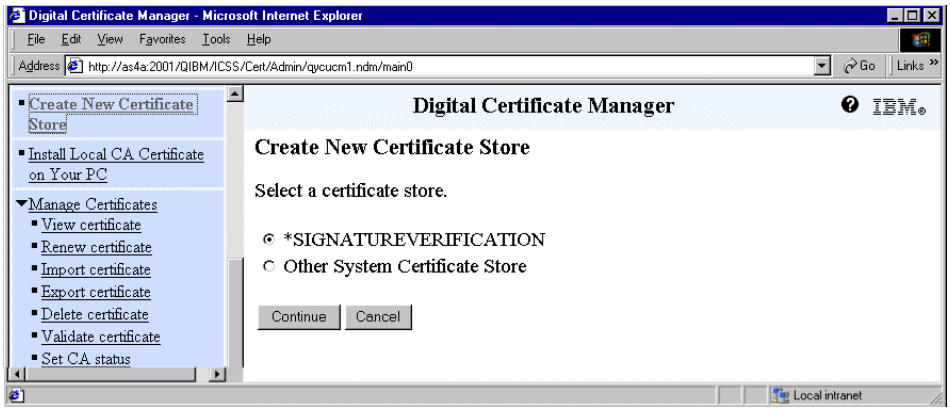


Figure 78. Create New Certificate Store

2. Select ***SIGNATUREVERIFICATION** and click **Continue**.

Figure 79. Copy Signing Certificates window

The Copy Signing Certificate window gives you the option to select whether you want to copy all existing object signing certificates on this system into the new signature verification store.

3. Select ***Yes** and click **Continue**.

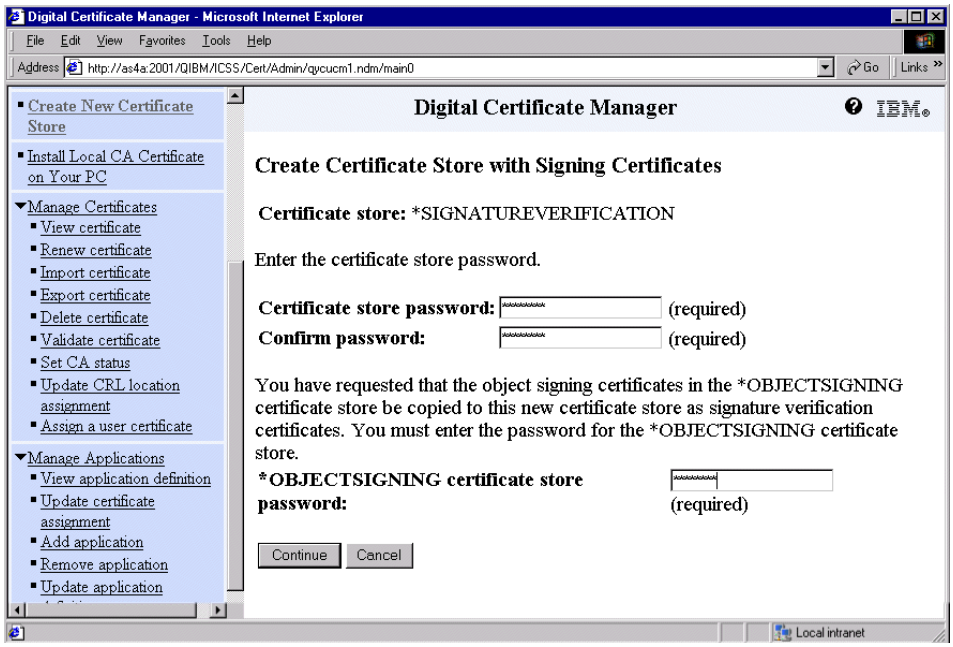


Figure 80. Create Certificate Store with Signing Certificates window

This option automatically opens the object signing certificate store, retrieves all object signing certificates as signature verification certificates and stores them in the signature verification certificate store. Note that the difference between an object signing certificate and a signature verification certificate is that the private key of the object signing certificate is not available, in fact not needed, for signature verification and therefore not exported.

4. Specify a new password for the signature verification store and enter the password of the existing object signing store and click **Continue**.

   A message is displayed confirming that the signature verification store has been created in the /QIBM/UserData/ICSS/Cert/Signing/VFYOBJ.KDB directory.

5. Click **OK** to continue.

The object signing certificate store is still open. In the next step the object signing certificate will be exported as a signature verification certificate into a file. This file can then be shipped along with the application package to customers.

6. Click **Export certificate** from the Manage Certificates menu in the navigation pane.

Figure 81. Export Certificate window

7. Select **Object Signing** and click **Continue**.

   In case you use a private or local Certificate Authority to sign objects, you also need to export the CA certificate of the CA that issued the object signing certificate.

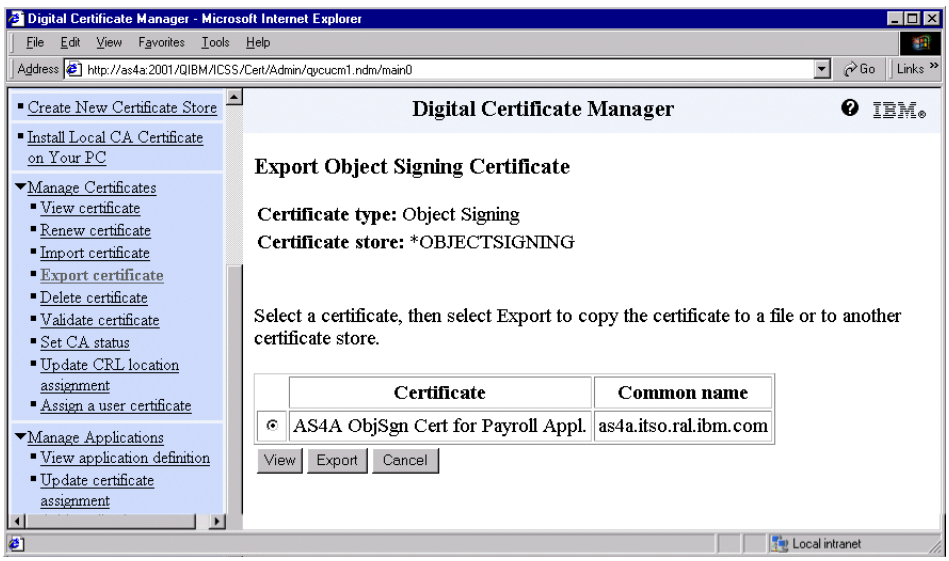


Figure 82. Export Object Signing Certificate window

8. Select the object signing certificate to be exported and click **Export**.

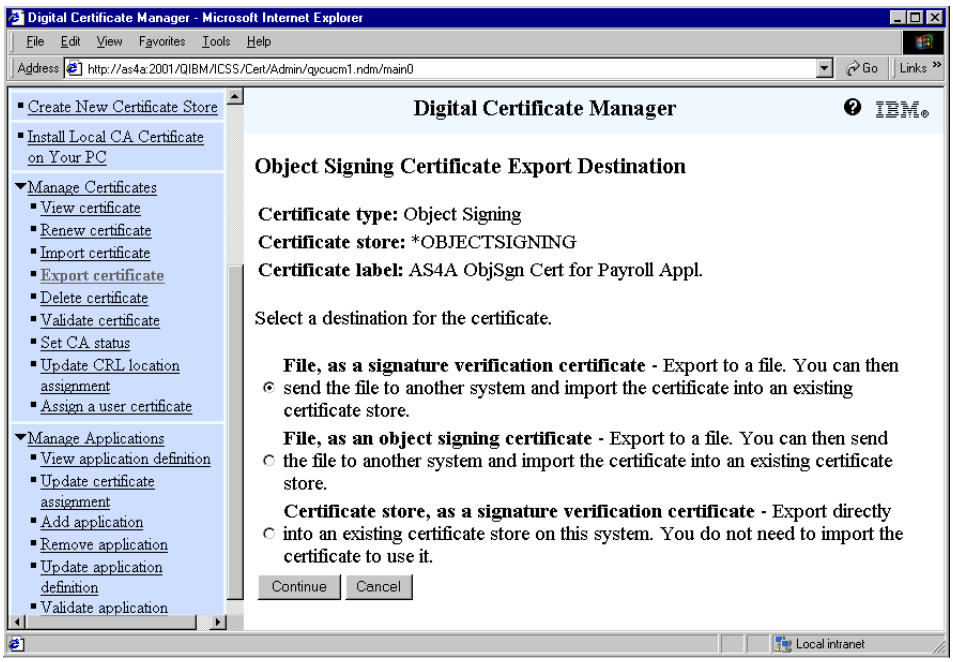


Figure 83. Object Signing Certificate Export Destination window

Select a destination for the certificate, as follows:

File, as a signature verification certificate Select this to export a copy of the selected certificate without its associated private key into a file for use on another iSeries or AS/400 server. The certificate can be used to verify the authenticity of objects that this object signing certificate signs.

File, as an object signing certificate Select this to export a copy of the selected certificate and associated private key into a file for use on another iSeries or AS/400 server. You can then assign applications to use this certificate to digitally sign objects on the other system. You can also use this option to create a backup copy of the object signing certificate.

Certificate store, as a signature verification certificate Select this to export a copy of the selected certificate into the *SIGNATUREVERIFICATION certificate store on this system. You can then use the certificate to verify the authenticity of objects that the corresponding object signing certificate signs. The *SIGNATUREVERIFICATION certificate store must exist before you can perform this action.

9. Select **File, as a signature verification certificate** and click **Continue**.

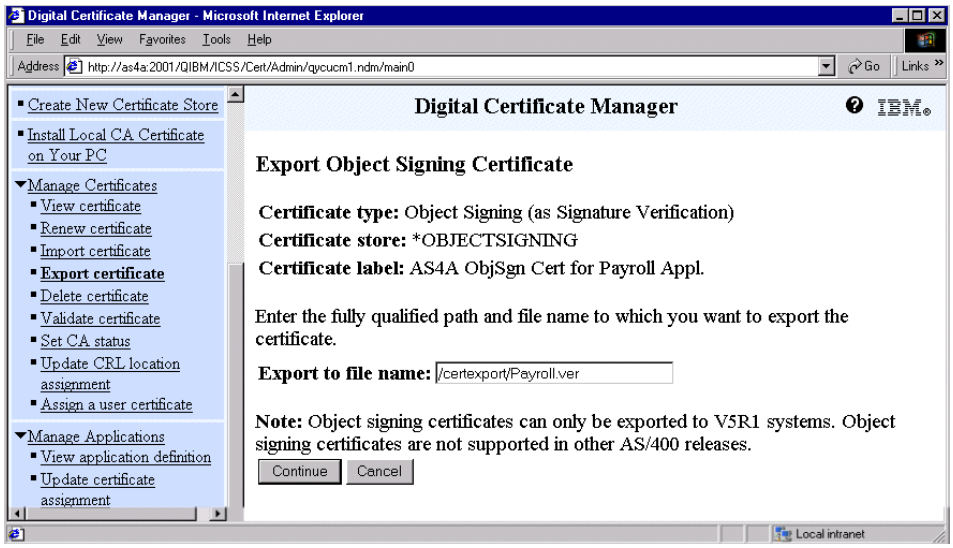


Figure 84. Export Object Signing Certificate window

10. Specify a path in an IFS directory to export the object signing certificate to. The file specified will the contain the signature verification certificate of the corresponding object signing certificate. For example:

    `/certexport/Payroll.ver`

11. Click **Continue** to perform the export.

    A confirmation message is displayed.

You completed the necessary steps to be able to start signing objects and verifying signatures on the development system. However, as mentioned in 3.3.2, “Scenario environment” on page 114, the person who is in charge of the Payroll application packaging should only be able to sign Payroll application objects. This person should not be able to sign objects from other application packages.

3.3.5 Authorizing users to use object signing applications

A goal in system and application security should always be that you limit access to functions and data to only those people who need to access them. In this scenario the system security officer has set up the object signing and signature verification environment. However, the person’s user profile who is in charge of signing application objects does not necessarily have *ALLOBJ and *SECADM special authorities. In fact, the person does not need this kind of authority when signing objects. The only authorizations this person needs are:

- *ALL object authority to all objects that make up the application
- Access to the object signing application to perform object signing
- *AUDIT special authorities to verify the signatures on the application objects

Perform the following steps to authorize the payroll application administrator MARION to sign objects of the payroll application:

1. Start **Operations Navigator**.
2. Select the development system and sign on with a user that has *ALLOBJ and *SECADM special authorities.
3. Right-click the development system.

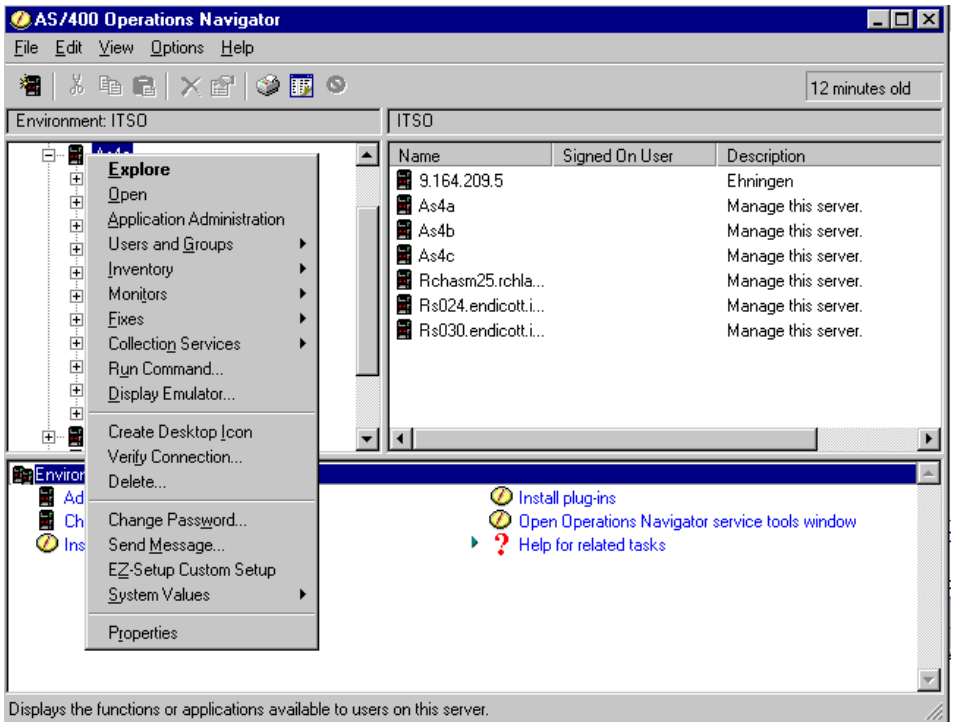


Figure 85. Operations Navigator - System context menu

4. Select **Application Administration** from the context menu.
5. Click the **Host Applications** tab and expand the **Digital Certificate Manager (DCM)** options.
6. Expand **Object Signing Applications**.

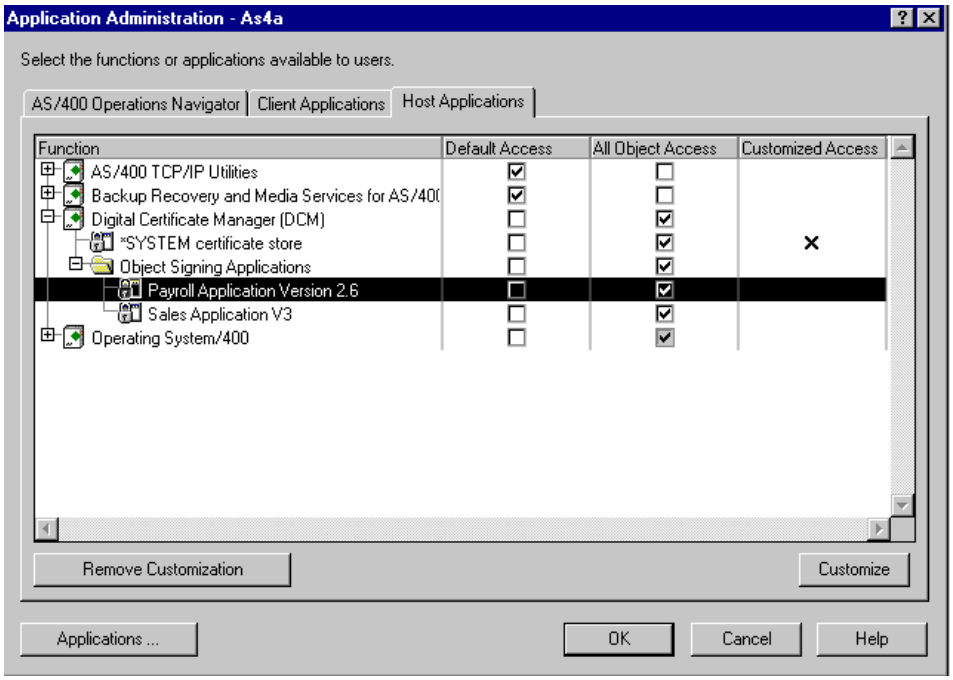


Figure 86. Operations Navigator - Application Administration - Host Applications

All configured object signing applications are displayed. The columns to the right of the object signing applications have the following meaning:

Default Access The settings in this column define whether users or groups can access the functions without being explicitly allowed or denied through the customize option. For security-related tasks and functions, you should not enable default access.

All Object Access With this setting you can specify if a user profile that has *ALLOBJ special authorities and no explicit permission can access the function. In other words, even if a user profile has *ALLOBJ, which means access to all objects on the system, you can specify that these users cannot perform this function. The customize option can still be used to grant access to an individual user or group.

Customized Access An X in this column means that the access list for this function has been modified. It could be that

users or group have been explicitly granted access or denied access.

7. Select the object signing application for which you want to modify access control (in this scenario it is the **Payroll Application Version 2.6**), and click **Customize** in the lower left corner of the window.

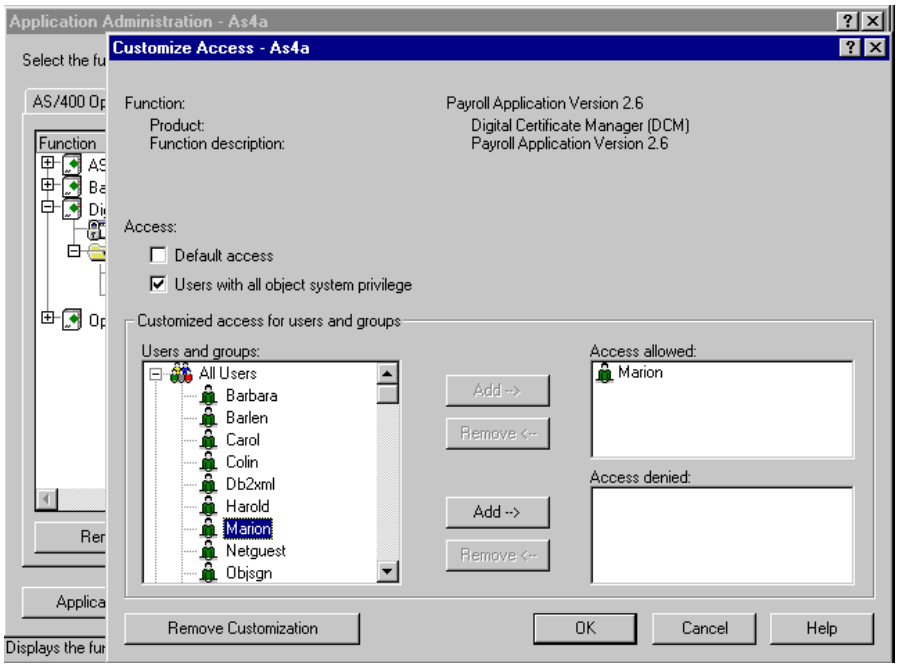


Figure 87. Operations Navigator - Customize Access window

The Customize Access window allows you to explicitly grant or deny access for the selected function to individual users or group. Note that you can also modify the Default access and the All object access from this window.

8. Select the payroll application administrator user **Marion** and click **Add** beside the Access allowed pane to allow user Marion to sign payroll application objects.
9. Click **OK** to save your changes.
10. Click **OK** in the Application Administration window to return to the Operations Navigator main window.

All setup tasks on the application development system AS4A are completed.

3.3.6 Signing the application objects using DCM

DCM is used to create the object signing certificate store for holding certificates that are used to sign objects. The DCM tasks for this certificate store also allow you to authenticate digital signatures on objects. An object signing certificate is used to digitally sign an object to ensure the integrity of the object itself and the origination of that object.

To sign an object, the private key of an object signing certificate along with a hash algorithm is used to create the signature. The receiver of the object must have access to a copy of the corresponding signature verification certificate in order to properly authenticate the object signature.

Signing an object can be performed in two different ways:

- Using the DCM interface as described in this section.
- Using an own application that uses the QydoSignObject API, as described in 3.4, “Object signing application programming interfaces” on page 176.

Let us suppose the programmers of Smith and Johnes Ltd. have completed their work and the payroll application is ready to be shipped. Marion, as the application administrator, now has to sign all relevant application objects as listed in Figure 62 on page 114. The required authorizations have been granted to Marion as follows:

- The user profile Marion has *ALL object authorities to all objects the payroll application consists of.
- The user profile also has *AUDIT special authorities to allow Marion to verify the signatures before saving the application package.
- User profile Marion has been granted access to the object signing application, Payroll Application Version 2.6, via Operations Navigator.

The following steps are performed by Marion to sign all objects:

1. Start the AS/400 Tasks page by using a Web browser and enter the URL:

   `http://servername:2001.`

   When prompted sign on with the application administrator’s user profile Marion and her password.

2. Click **Digital Certificate Manage**r from the AS/400 Tasks page.

   DCM starts and shows all available tasks for user Marion.

3. Click **Manage Signable Objects** on the navigation pane to expand its options.

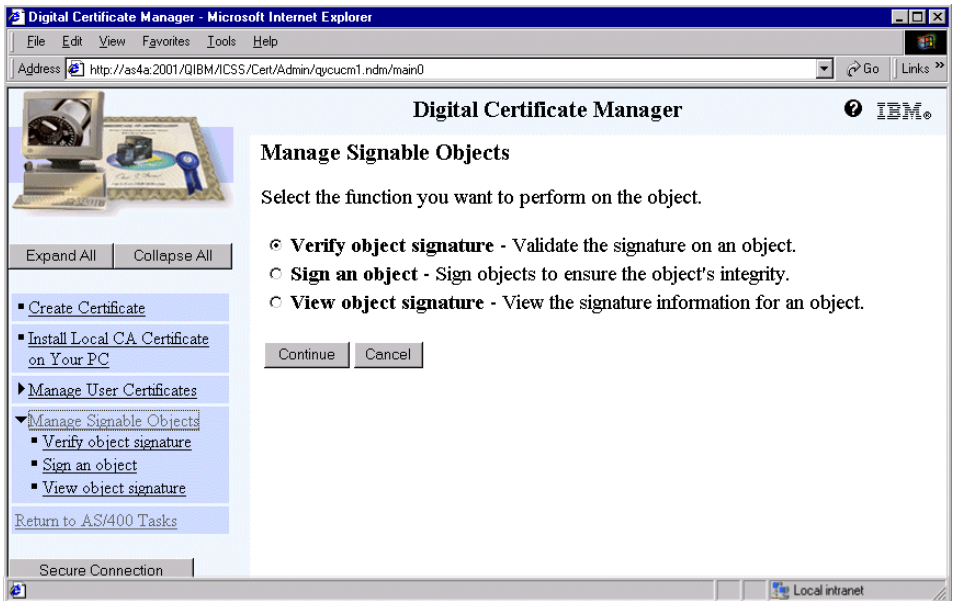


Figure 88. Manage Signable Objects window

Note that the user profile Marion gets only a limited number of options displayed in the navigation pane. One of the major differences from a user with *SECADM and *ALLOBJ is that this user does not need to open a certificate store.

A user profile with no *AUDIT special authorities and no authority to use an object signing application will only see the View object signature option. If *AUDIT is defined for the user, the Verify object signature option is also shown. The *Sign an object* option is only displayed when the user profile signed on to DCM is authorized to use at least one object signing application.

The options on the Manage Signable Objects window are as follows:

Verify object signature Select this to authenticate a digital signature on an object.This function requires that the object signing certificate has been exported to the signature verification store.

Sign and object Select this to use an application and its assigned certificate to digitally sign an object to ensure the object's integrity.

View an object signature Select this to view the signature information for a digitally signed object.

4. From the list of tasks, select **Sign an object** and click **Continue** to display a list of application definitions that you can use for signing objects.

   Since user Marion is authorized to use only the Payroll Application Version 2.6 no other object signing application is displayed.

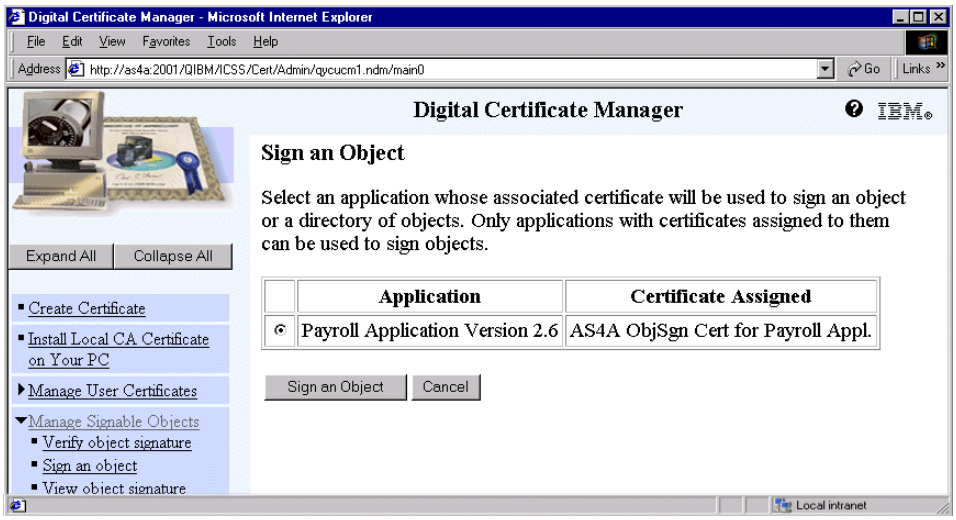


Figure 89. Sign an Object window: Application selection

An application ID controls how much authority is required for someone to sign an object with a specific certificate and provides another level of access control beyond that which DCM provides. It also describes the type of objects that you plan to sign with a specific certificate. The application definition requires that a user either has *ALLOBJ special authority or is explicitly authorized via Operations Navigator to use the certificate for the application to sign objects.

Note

The list shows only those object signing applications a user is authorized to use and a certificate has been assigned to.

5. Select the relevant application, in this case the Payroll Application Version 2.6, and click **Sign an Object** to display the window in Figure 90 on page 144 for specifying the location of the objects that you want to sign.

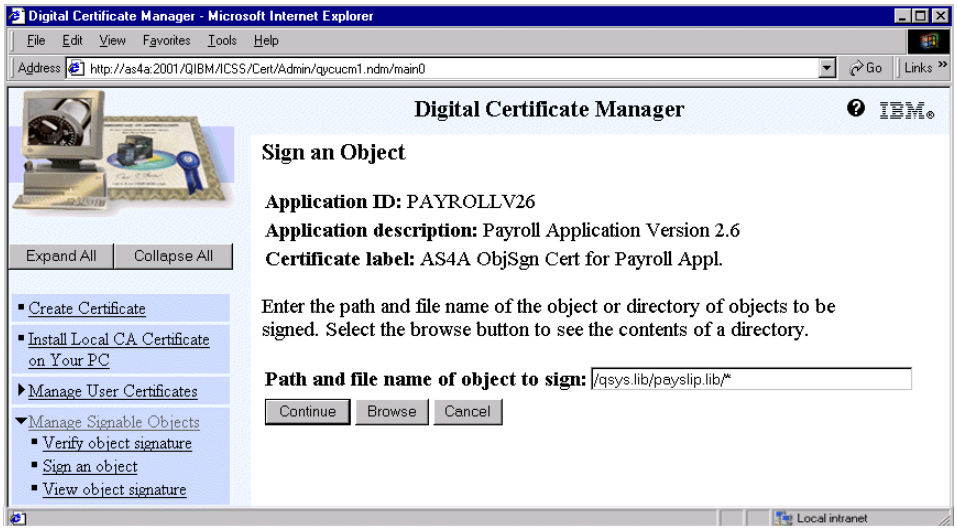


Figure 90. Sign an Object window - object selection

There are several ways to specify the location or select the object or objects that you want to sign.

- You could enter the fully qualified path and file name of the object or objects within a directory or library that you want to sign.
- You could enter a directory location, for example /payfiles/*, and click **Browse** to view the contents of the directory and select an object for signing.
- You could use certain wildcard characters to describe the part of the directory that you want to sign. The asterisk (*) specifies “any number of characters” and the question mark (?) specifies “any single character”.

The object signing for the payroll application has to be performed in several steps because the affected objects are located in different libraries and directories.

The administrator has decided to start signing all signable objects in the library PAYSLIP. The path and object names have to be specified using the IFS name format beginning with a leading slash. Omitting the leading slash results in an error.

6. Enter the path information `/qsys.lib/payslip.lib/*` and click **Continue** to display the Processing Options for Signing an Object window. The * in the path name specifies that all objects in library PAYSLIP are processed.

Note that only signable objects as specified in 3.1.3, “Objects that can be signed” on page 102 are processed.

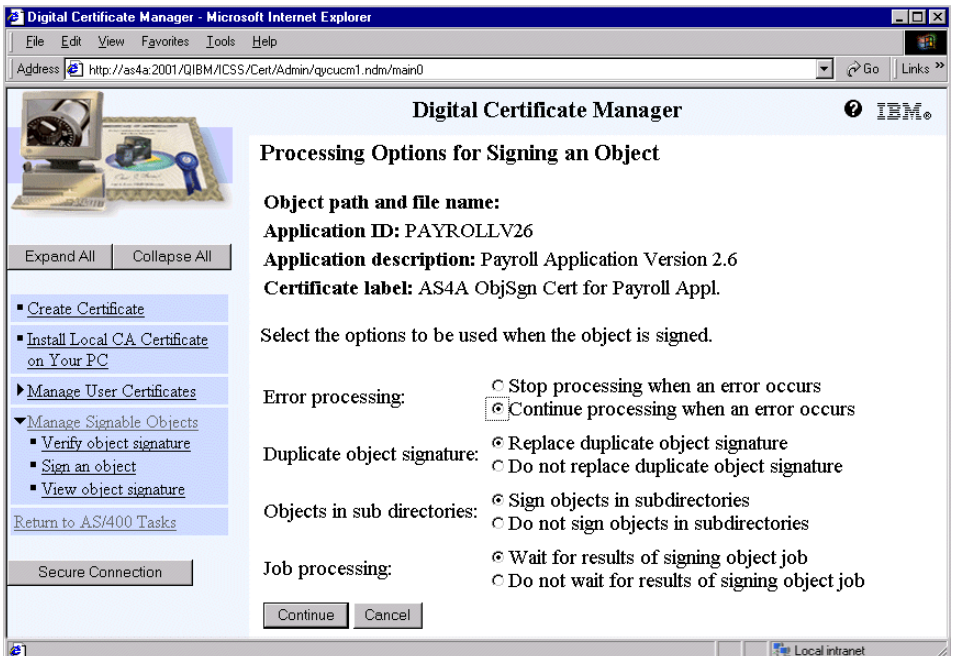


Figure 91. Processing Options for Signing an Object window

You have several choices to process object signing:

Stop processing when an error occurs This will stop the object signing process when an error occurs. If the signing process for one object in the sequence fails, the signing process stops without creating signatures on any remaining objects.

Continue processing when an error occurs This option will ignore any errors and continue the object signing process that attempts to sign all objects in this job. Job results are sent to the file specified. You would want to choose the option to continue because of the possibility of empty save files, which will cause an error. However, you should still check the job log to ensure there are no other errors.

Replace duplicate object signature The application will replace the original signature, if it is a duplicate signature, with a new signature.

Do not replace duplicate object signature The original signature will be left in place and the application will continue processing.

Sign objects in subdirectories Select this if you want to individually sign objects in any subdirectories found in the path that you specified.

Do not sign objects in sub directories This will sign only those objects in the main directory that you specified in the path and file name, but will not individually sign each object within any subdirectories.

Wait for results of signing object job Select this if you want DCM to wait until the object signing process finishes for all objects before displaying signing results. If you are only signing one or a couple of objects, then you may want this option as results are shown directly in the window.

Do not wait for results of signing object job DCM will run the object signing job in batch mode and writes the results of the job to a file. You can then check the job results at a later time.

7. Select **Continue processing when an error occurs** and leave the remaining options to their default. Click **Continue** to display the Job Results File for Signing an Object window.

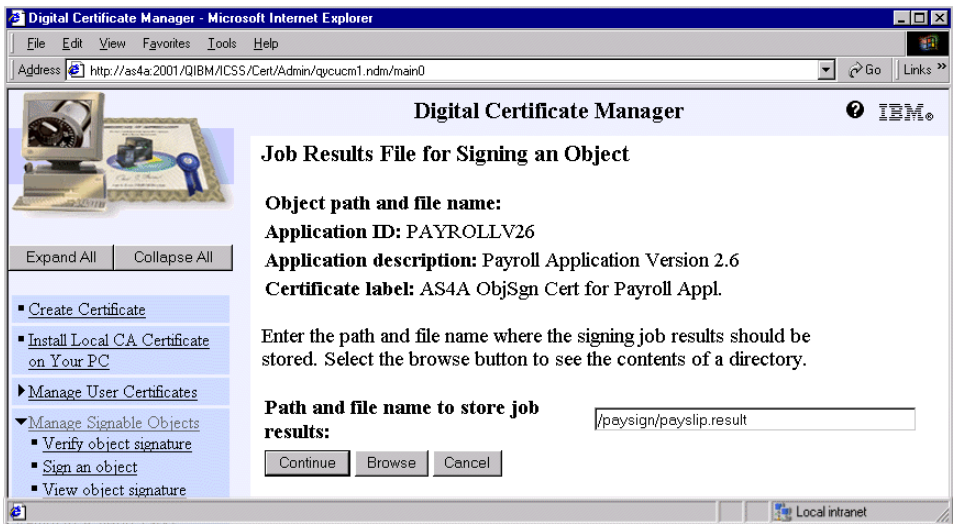


Figure 92. Job Results File for Signing an Object window

8. Enter the path and file name /paysign/payslip.results where the signing job results should be stored. Note that the paysign directory already exists and will be used for all object signing and signature verification tasks.

   Path and file name to store job results Enter the fully qualified path and file name to use for storing job results for the object signing operation.

   Browse Click this to view the contents of the directory that you specified in the Path and file name to store job results field. You can then select an existing file from that directory for storing job results. The new results will be added to existing ones. If the specified path is empty, you receive a directory browsing error.

   Results files cannot reside under /QSYS.LIB and are required to have a CCSID of 13488.

9. Click **Continue**.

   Since you have selected the processing option **Wait for results of signing object job**, you will get the following window:

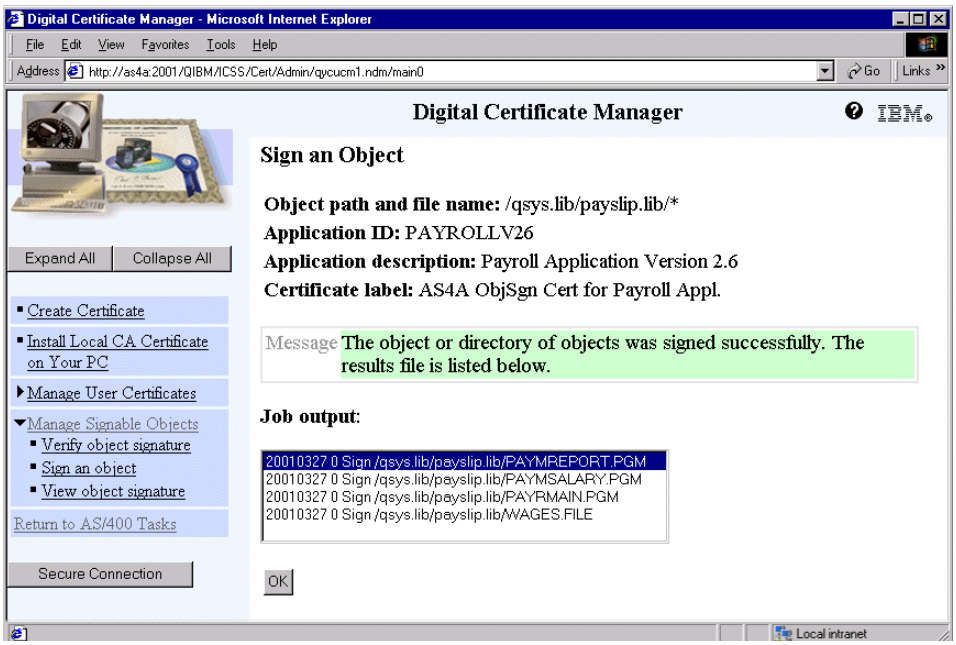


Figure 93. Sign an Object window: Signing results

The job results are displayed in DCM.

10. Click **OK** to return to the Manage Signable Objects window.

    You can also look at the entries in the job results file at a later time by entering the following command from an OS/400 command line:

    ```
    DSPF STMF('/paysign/payslip.result')
    ```

    This command displays the results file as specified in Figure 94.

```
Browse : /paysign/payslip.result
Record :       1   of       4 by  18                    Column :    1     92 by 131
Control :

....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9...
********Beginning of data**************
          20010327        0 Sign                 /qsys.lib/payslip.lib/PAYMREPORT.PGM
          20010327        0 Sign                 /qsys.lib/payslip.lib/PAYMSALARY.PGM
          20010327        0 Sign                 /qsys.lib/payslip.lib/PAYRMAIN.PGM
          20010327        0 Sign                 /qsys.lib/payslip.lib/WAGES.FILE
********End of Data********************



 F3=Exit   F10=Display Hex   F12=Cancel   F15=Services   F16=Repeat find   F19=Left   F2(
            (C) COPYRIGHT IBM CORP. 1980,  2000.
```

Figure 94. Results when object signing done in batch

The messages shown in the results file indicate that all objects have been signed successfully.

Next the administrator will sign the single file insopt.cfg in directory payins1. The options taken in the following steps include:

- Browsing a directory for objects to sign
- Perform object signing in batch mode

11. Select **Sign an object** from the Manage Signable Objects window and click **Continue**.

12. Select **Payroll Application Version 2.6** and click **Sign an object**.

Table 3. Sign an Object window: Path selection

As previously described, you can sign objects by selecting them in different ways. You have already seen how to sign objects using the wildcard character '*' and clicking **Continue**. This time you browse for a specific file in the directory payins1.

13. Enter `/payins1/*` for the path and file name and click **Browse**.

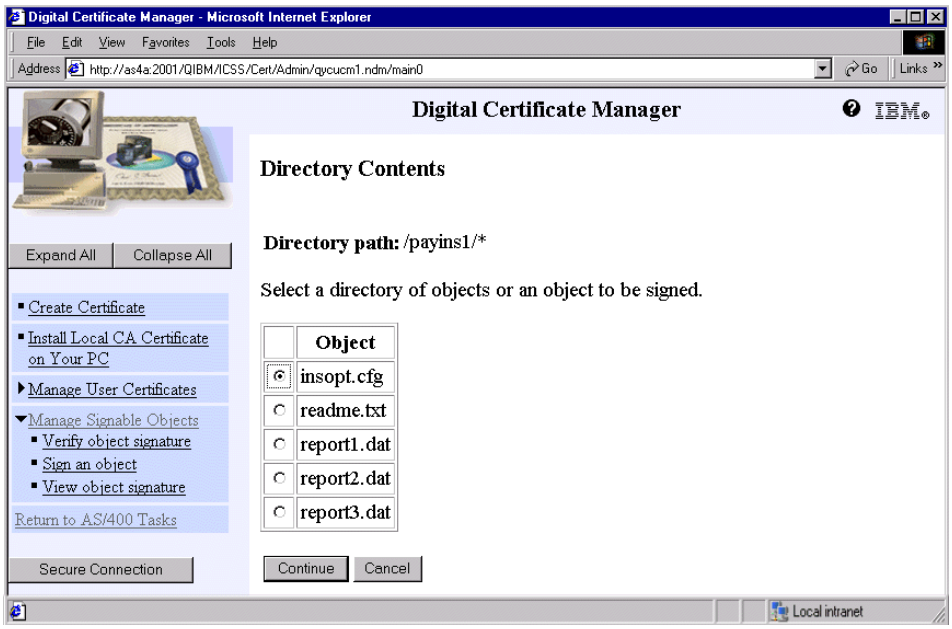


Figure 95. Directory Contents window

All files in the specified directory are displayed. You can only select one file from this list. Multiple selections are not possible.

14. Select the **insopt.cfg** file and click **Continue**. You return to the previous window, but this time the entire path /payins1/insopt.cfg is shown in the path and file name field.
15. Click **Continue** to proceed to the Processing Options for Signing an Object window.
16. Select **Do not wait for results of signing object job** for the job processing option and leave the remaining options to their default.

    Select the processing option **Do not wait for result of signing object job** if you want DCM to run the object signing job in batch mode. You can then check the job results file at a later time. Selecting this option is useful when you are signing a large number of objects, since you can still use DCM for other tasks while the signing process runs and completes.

    Another advantage of signing objects in batch mode is that you have one joblog per object signing job as opposed to signing objects in DCM where you have to look through all ADMIN HTTP server jobs to find the correct one.
17. Click **Continue** to display the Job Results File for Signing an Object window.
18. On the Path and file name to store job results parameter, specify `/paysign/payins1.results`.
19. Click **Continue**.
20. Click **OK** to return to the Manage Signable Objects window.

    The message in Figure 96 indicates that the job was submitted to sign objects. Look at the QOBJSGNBAT joblog for user Marion to view the job results. The QOBJSGNBAT job is submitted by using the QDFTJOBD job description. By default, a batch job using this job description does not create a joblog spool file when the job completed normal. In all other cases you have a joblog that contains very good information about the errors that occurred. The following example shows an object signing process that failed because of lack of authority to the object to be signed.

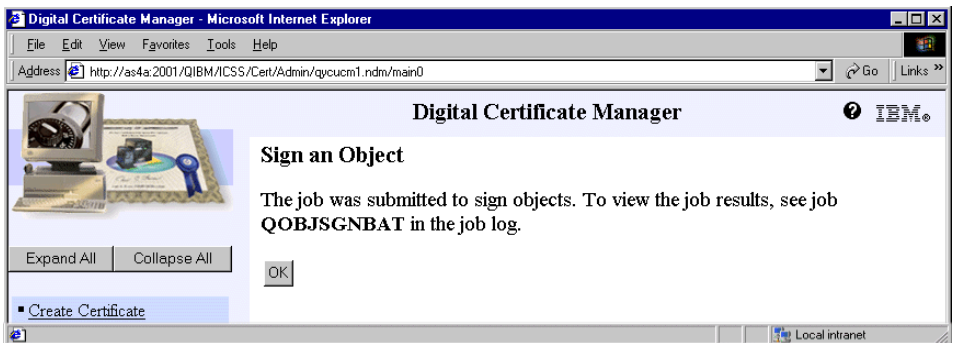


Figure 96. Sign an Object window: Job log information

```
                                                Display Spooled File
 File  . . . . . :   QPJOBLOG                                                    Page/Line
 Control . . . . .                                                               Columns
 Find  . . . . . .

*...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9....+....0....+.
..2....+.
                                Message . . . . :  -QSYS/CALL PGM(QICSS/QYCUDRIVER) PARM('92' ''
                                 '/payins1/insopt.cfg' 'PAYROLLV26' '1' '0' '1' '/paysign/payins1.results'
 MCH1001    Escape              40    03/27/01   17:47:19   #auexcpt                 000AEC   QYDO_UTIL
*
                                 To module . . . . . . . . . :   QYDOCMI
                                To procedure  . . . . . . . :   CipherMI__11YdoCipherMIFv
                                 Statement . . . . . . . . . :   14
                                Message . . . . :   Attempt to use permanent system object *N without
                                    authority.
                                Cause . . . . . :   You tried to use the permanent system object *N without
                                   having the correct authority.
 CPFB749    Escape              30    03/27/01   17:47:19   QYDO_UTIL     QSYS       *STMT    QYCUYDOMGR
*
                                 From module . . . . . . . . :   QYDO_UTIL
                                 From procedure  . . . . . . :   qydo_SendMessage
                                 Statement . . . . . . . . . :   1832
                                 To module . . . . . . . . . :   QYCUYDOMGR
                               To procedure  . . . . . . . :   ycu_signGivenObjects__FPCciPPcP6Qus_ECPC16q
                                    _ErrorHandle
                                 Statement . . . . . . . . . :   1933
                               Message . . . . :   Object signature operation ended abnormally. 1 objects

 F3=Exit   F12=Cancel   F19=Left   F20=Right   F24=More keys
```

During the last steps of signing the payroll application, the administrator again uses a different way of selecting the objects to be signed.

21. Select **Sign an object** from the Manage Signable Objects window and click **Continue**.

22. Select **Payroll Application Version 2.6** and click **Sign an object**.

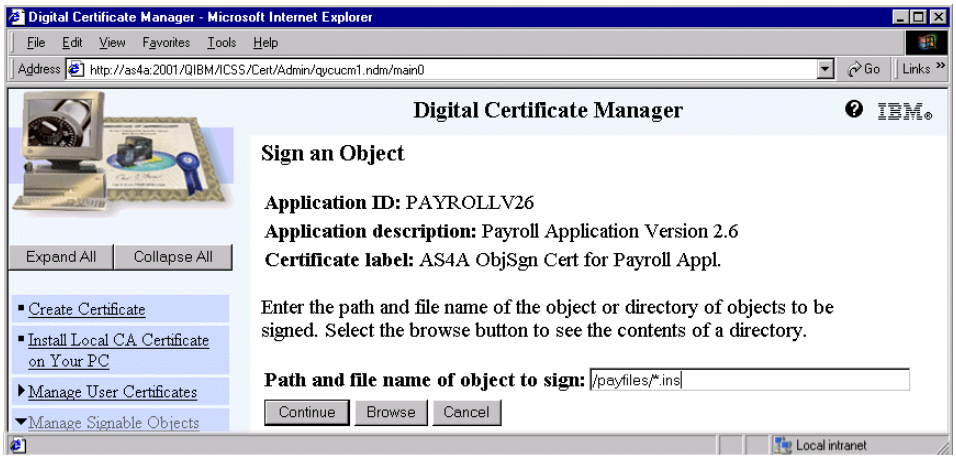


Figure 97. Sign an Object window: Path selection

In the third part of the signing process, the payroll applications administrator needs to sign all objects in the payfiles directory that have an extension of .ins.

23. Enter `/payfiles/*.ins` for the path and file name and click **Continue** to display the Processing Options for Signing an Object window.
24. Accept the defaults on the Processing Options for Signing an Object window and click **Continue**.
25. In the path and file name for the results file, enter `/paysign/payfiles.result` and click **Continue**.

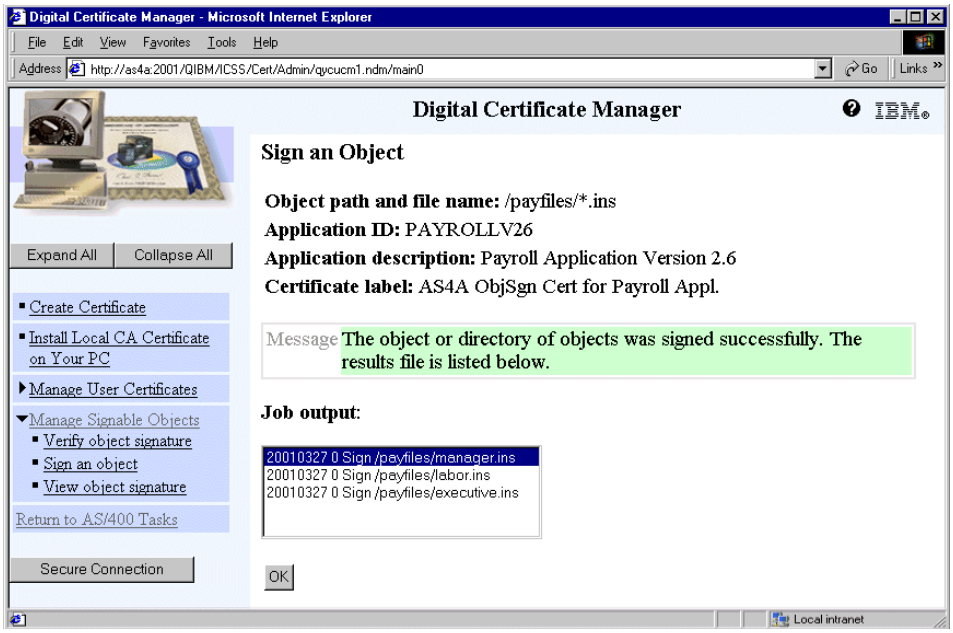


Figure 98. Sign an Object window: Signing results

With the /payfiles/*.ins file selection all files with an .ins extension have been signed as shown in the results file in Figure 98.

At the time this redbook was written, we experienced the following errors:

- Signing an empty save file

  If you try to sign an empty save file, it will result in a failure. The results file will contain the following information:

  ```
  CPFB74C   20010214      0 Sign          /qsys.lib/vanessa.lib/empty.file
  ```

  In this case this is an error saying that the object was not signed, because it was an empty save file.

  The DCM interface presented a general error message and advised us to look into the ADMIN HTTP server joblog for error details.

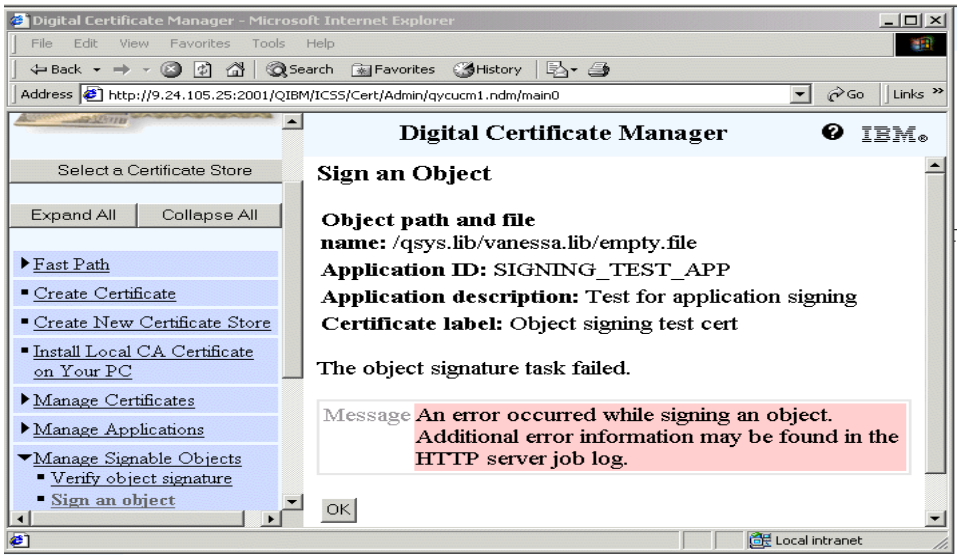


Figure 99. Sign an Object window: Trying to sign an empty save file

The ADMIN HTTP joblog shows message CPFB74C.

```
Message ID . . . . . . :   CPFB74C       Severity . . . . . . . :   40
Message type . . . . . :   Escape
Date sent  . . . . . . :   03/27/01      Time sent  . . . . . . :   18:38:40

Message . . . . :   Cannot sign an empty object.
Cause . . . . . :   The object was empty and cannot be signed. The object was
  /qsys.lib/vanessa.lib/empty.file.
Recovery  . . . :   If the object is a savefile, save objects into the
  savefile and retry the operation.
Technical description . . . . . . . . :   The return code was 202. The object
  was /qsys.lib/vanessa.lib/empty.file. The certificate label, if any, was .
```

- Signing an object from a previous release

  Signing an object on a V5R1 system that was compiled for a previous release, for example V4R5, will fail, because releases prior to V5R1 would not recognize the object's format.

  The same error message as shown in Figure 99 is also shown for this situation. The ADMIN HTTP server joblog shows the following message:

```
Message ID . . . . . .  :   CPFB721      Severity . . . . . . .  :   40
Message type . . . . .  :   Escape
Date sent  . . . . . .  :   03/27/01     Time sent  . . . . . .  :   18:34:06

Message . . . . :   Object has a target release that prevents it from being
  signed.
Cause . . . . . :   Object signing support was not available on the release
  specified as the target release for the object. Therefore the request to
  sign this object is not valid. The object was /qsys.lib/barlen.lib/v4r5.pgm.
Recovery  . . . :   Recreate the object using a TGTRLS which supports digital
  signatures.  For *PGM, *MODULE, and *SRVPGM, digital signing support begins
  in V5R1M0.
Technical description . . . . . . . :   The return code was 300. The
  certificate information, if any, was .
```

The results file contains the message ID too:

```
CPFB721      20010327     0 Sign       /qsys.lib/barlen.lib/v4r5.pgm
```

3.3.7 Packaging the application

Before the signed objects are saved to a save file and sent to the customer system, it is advisable to verify their signatures to ensure no integrity violations have occurred and that the objects are still signed. In a typical environment it is not unusual that a programmer makes some last-minute changes and re-compiles one of the program objects. Re-compiling, for example, deletes the current object and creates a new one. With the compilation the signature is also gone and the object needs to be signed again. Note that saving the objects to tape works just as well as using save files.

Verifying signatures on the development system requires you to have a signature verification store with the proper signature verification certificate in place. This step was already performed in 3.3.4, “Exporting object signing certificates for verification” on page 131. To verify object signatures you can use three different methods:

- Using the Check Object Integrity (`CHKOBJITG`) command as described in 3.2.3, “Check Object Integrity (CHKOBJITG) command” on page 108.
- Using the DCM interface as described later in 3.3.10, “Using DCM to verify object signatures” on page 166.
- Using an application that uses the QydoVerifyObject API, as described in 3.4, “Object signing application programming interfaces” on page 176.

Another way of verifying object signatures is during restore of objects. In this case the signature verification is done automatically, depending on the setting of the system value QVFYOBJRST.

In this scenario the payroll application administrator is using the CHKOBJITG command. All signed objects belonging to the application that is to be shipped are owned by user PAYROLL.

To check the signature for objects that were initially signed, the object selection criteria by owner profile does not work. Using the USRPRF parameter to check signatures for all objects owned by PAYROLL would also list other objects that are signable, but were intentionally not signed.

In this case the administrator performs the following three commands to verify the signatures of all affected objects:

```
CHKOBJITG OBJ('/qsys.lib/payslip.lib/*') OUTFILE(MARION/PAYSIGN)
    CHKSIG(*ALL)
CHKOBJITG OBJ('/payins1/insopt.cfg') OUTFILE(MARION/PAYSIGN)
    OUTMBR(*FIRST *ADD) CHKSIG(*ALL)
CHKOBJITG OBJ('/payfiles/*.ins') OUTFILE(MARION/PAYSIGN)
    OUTMBR(*FIRST *ADD) CHKSIG(*ALL)
```

The first of the above commands places the results in a file named PAYSIGN in library MARION. If the file already exists, the command will replace the current entries. The second and third command will add their results to the PAYSIGN files without replacing existing entries.

In the next step, the administrator displays the results file with the following command:

```
DSPPFM FILE(MARION/PAYSIGN)
```

The results file is displayed.

```
                    Display Physical File Member
File . . . . . . :  PAYSIGN         Library . . . . :  MARION
Member . . . . . : PAYSIGN          Record . . . . . :  1
Control  . . . . .                 Column . . . . . :  1
Find . . . . . . .
*...+....1....+....2..//..+....5....+....6....+..//+....2....+....
1032701193742AS4A    1   *PGM    PAYROLL   NOSIG  /qsys.lib/payslip.lib/PAYMREPORT.PGM
                    ****** END OF DATA ******
```

Note that the output as displayed in this section has been modified to fit on the page.

As almost expected, a programmer has changed a program object. The result file lists all objects as invalid and no signatures (CHKSIG(*ALL)). In this case

the NOSIG value indicates that the object PAYMREPORT.PGM in library PAYSLIP has no signature.

The administrator needs to sign this object again, as described in 3.3.6, “Signing the application objects using DCM” on page 141.

When verification has completed successfully, save the signed objects, whether individual or many objects in a library, into a save file. This package can be shipped to the customer systems.

Note

To ensure integrity of objects that are not signable, for example physical files, you can first save all objects into a save file and then sign the save file.

When you send the signed objects to others, you must include a copy of the certificate that signed the objects. This is done by using DCM to export the object signing certificate (without the certificate’s private key) as a signature verification certificate. The export function for the payroll application was already performed in 3.3.4, “Exporting object signing certificates for verification” on page 131.

In this scenario, the payroll application administrator has saved all objects from the payins1 and payfiles directories into a save file PAYIFSAPP in library PAYSLIP using the command:

```
SAV DEV('/qsys.lib/payslip.lib/payifsapp.file') OBJ(('/payfiles/*')
   ('/payins1/*'))
```

Then the library is saved into a save file PAYRAPP into library PAYSHIP using the command:

```
SAVLIB LIB(PAYSLIP) DEV(*SAVF) SAVF(PAYSHIP/PAYRAPP)
```

To ensure the integrity of all the objects within the save file PAYRAPP (including the unsigned objects), the application administrator signs the application save file by performing the steps described in 3.3.6, “Signing the application objects using DCM” on page 141.

You can send the signed objects to the target systems in a save file using FTP and the `PUT` subcommand. You could also save the signed objects onto tape and restore onto the customer’s system. Or you could send the save file using the `SNDNETF` command. In addition to shipping the application to the remote system, you must also send a copy of the signature verification

certificate. In case you use an object signing certificate issued by a private CA, you also need to send a copy of the CA certificate.

Bear in mind some of your customers may have AS/400s at release of operating system prior to V5R1. Signed objects cannot be shipped to systems at a lower level of OS/400. When they are saved using the SAVOBJ command, with the target release TGTRLS parameter set to the relevant level of OS/400, the digital signature is stripped off the object. The objects can then be shipped to and restored on the remote AS/400 without problems.

The administrator finishes the process by FTPing the following files over to the customer system:

- Save file PAYRAPP in library PAYSHIP
- Signature verification certificate stored in file payroll.ver in directory certexport

The following is an extract of the FTP session used to transfer the files to the customer system WOLFPACK.

```
Previous FTP subcommands and messages:
  Server NAMEFMT is 1.
  Client NAMEFMT is 1.
> ascii
  200 Representation type is ASCII nonprint.
> put /certexport/payroll.ver /certimport/payroll.ver
  227 Entering Passive Mode (9,24,105,25,125,118).
  150 Sending file to /certimport/payroll.ver
  250 File transfer completed successfully.
     920 bytes transferred in 0.001 seconds. Transfer rate 942.080 KB/sec.
> bin
  200 Representation type is binary IMAGE.
> put /qsys.lib/payship.lib/payrapp.savf /qsys.lib/payrcv.lib/payrapp.savf
  227 Entering Passive Mode (9,24,105,25,150,121).
  150 Sending file to member PAYRAPP in file PAYRAPP in library PAYRCV.
  250 File transfer completed successfully.
  582912 bytes transferred in 9.229 seconds. Transfer rate 63.164 KB/sec.
```

3.3.8 Setting up a customer system for signature verification

When an object has been digitally signed, its digital signature can be verified by the user who receives it. DCM can be used to manage the signature verification certificates that validate digital signatures on objects.

3.3.8.1 Managing certificates for verifying object signatures

The private key of a certificate is used to create the signature on an object. When the signed object is sent to others, you must include a copy of the

certificate that signed the object. DCM exports the object signing certificate (without the certificate's private key) as a signature verification certificate.

To validate a signature on an object, you must have a copy of the certificate that signed it. You use the signing certificate's public key to examine and verify the signature that was created with the corresponding private key. You must also have a copy of the CA certificate of the CA that issued the certificate that signed the object. This is to verify the authenticity of the object signing certificate (exported as a signature verification certificate).

3.3.8.2 Prerequisites for signature verification

When objects are distributed, recipients must use a V5R1 version of DCM to validate the signature on the objects to ensure that the data is unchanged and to verify the identity of the sender.

To use DCM to verify object signatures, you must first create the *SIGNATUREVERIFICATION certificate store for managing the necessary signature verification certificates. When this is created, DCM automatically populates it with copies of most well-known public CA certificates.

To validate a signature, they need a copy of the signature verification certificate. A copy of this certificate was provided as part of the application package.

In addition the receiver must also have a copy of the CA certificate of the CA that issued the certificate used to sign the object. DCM provides copies of CA certificates from most well-known CAs.

If the object was signed with a certificate from a well-known Internet CA, as in this scenario, the receiver probably has a copy of the necessary CA certificate. If, however, you signed the objects with a certificate from a public CA that is not in the list of system-provided CAs or another private CA, you need to provide a copy of this, preferably sending it in a separate package, or make the CA certificate publicly available at the request of those who need it.

3.3.8.3 Verifying signatures signed on the same system

If you are verifying signatures for objects that were signed on the same system, you could perform signature verification while working within the objectsigning certificate store. But the *SIGNATUREVERIFICATION certificate store must still exist and must contain a copy of the certificate that signed the object. If verifying signatures that you create, you can export a signature verification certificate into the *SIGNATUREVERIFICATION certificate store.

To summarize requirements for customer system:

- The customer/recipients system must be at OS/400 V5R1.
- The signature verification certificate store must be created.
- Copy of CA certificate of the CA that issued the certificate.
- Copy of signature verification certificate.

Note

If you want to be able to verify signatures that you created with your own object signing certificate, you must create the *SIGNATUREVERIFICATION certificate store and copy the certificate from the *OBJECTSIGNING certificate store into it. This is true even if you plan to perform signature verification from within the *OBJECTSIGNING certificate store.

3.3.8.4 Setting system value QVFYOBJRST on a customer system

On the customer system you need to decide what value to set the QVFYOBJRST system value to. See 3.2.2, “QVFYOBJRST system value and restore operations” on page 107, for detailed information about the system value. For the payroll application, the system value is set to 3 due to the following reasons:

- The payroll application contains objects that are not signed, but need to be restored. These are objects that are not critical to the application, but required.
- No payroll application object with an invalid signature should be restored on the customer system. This is to ensure that the critical files have not been changed.

3.3.8.5 Setting up the signature verification store

Perform the following steps to set up the signature verification store on the customer system:

1. Use a Web browser and enter the following URL to display the AS/400 Tasks page:

   `http://servername:2001)`

   This requires that the HTTP ADMIN server instance is up and running. When prompted sign on with a user profile that has *ALLOBJ and *SECADM special authorities. Note that this authority level is required only to set up the signature verification environment. For verifying signatures

you can use a user profile that has access only to the objects to be checked and *AUDIT special authorities.

2. Click **Digital Certificate Manager** from the AS/400 Tasks page.
3. Click **Create New Certificate Store** in the left-hand navigation pane.

> **Note**
>
> If you have any questions about how to complete a specific form in this guided task, select the question mark (?) in the top left of window.

4. Select ***SIGNATUREVERIFICATION** as the certificate store to create and click **Continue**.

   If a *OBJECTSIGNING certificate store exists, DCM will prompt you to specify whether to copy the object signing certificates into the new certificate store as signature verification certificates as described in 3.3.4, “Exporting object signing certificates for verification” on page 131.

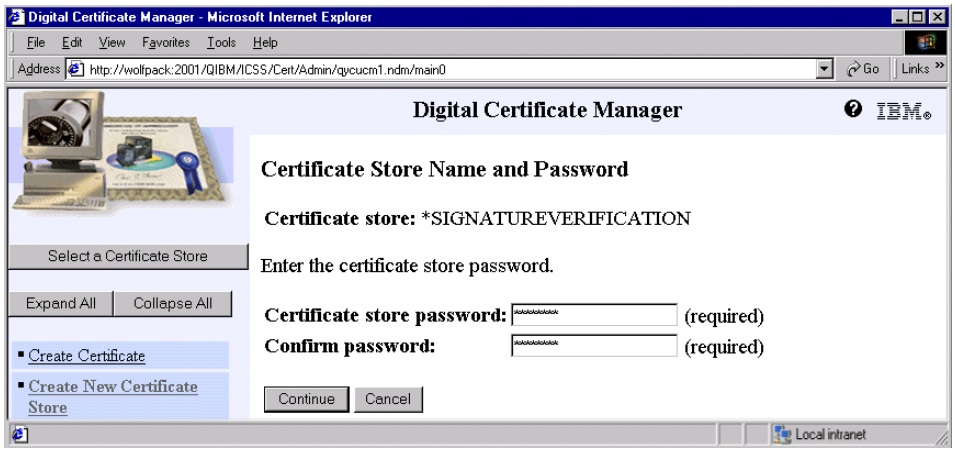


Figure 100. Certificate Store Name and Password window

5. Enter a certificate store password and confirm password and then click **Continue**.

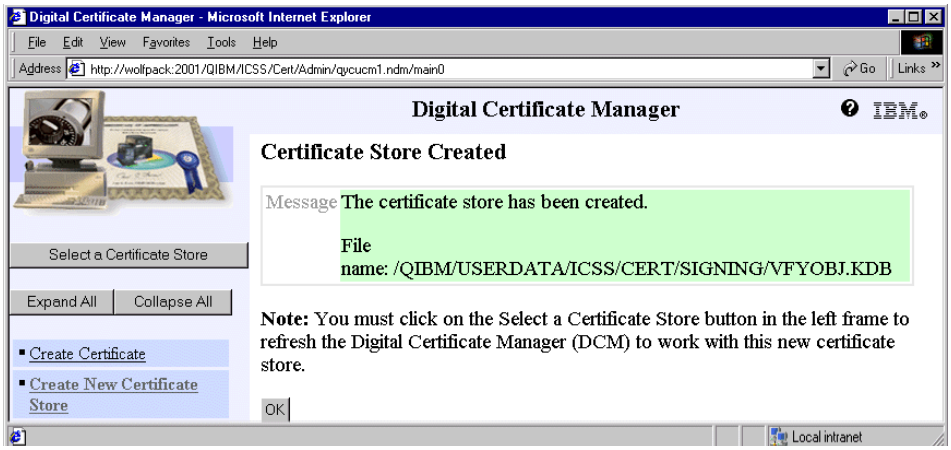


Figure 101. Certificate Store Created window

6. Now you can use the store to manage and use certificates to verify object signatures.

> **Note**
>
> If you created this store to verify signatures on objects that you signed, you do not need to complete the following steps. As new object signing certificates are created, export them from the *OBJECTSIGNING certificate store into this certificate store. If you do not export them, it will not be possible to verify signatures created with them.
>
> If the *SIGNATUREVERIFICATION certificate store is created to verify signatures on objects received from other sources, you should continue with the procedures below and import the required certificates into the certificate store.

7. In the navigation pane, click **Select a Certificate Store** and select ***SIGNATUREVERIFICATION** as the certificate store to open and click **Continue**.
8. When the password prompt is displayed, enter the certificate store password (the one you specified for the certificate store when you created it) and click **Continue**.
9. Click **Manage Certificates** in the left-hand navigation pane and select **Import Certificate**.

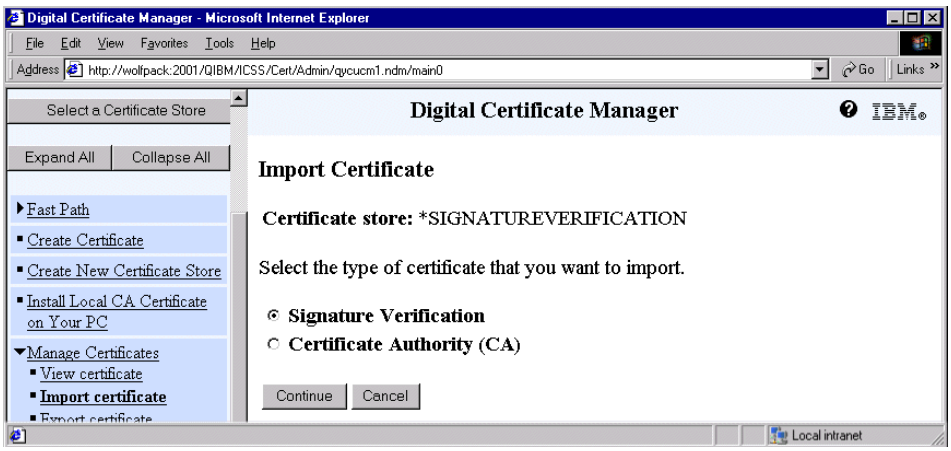


Figure 102. Import Certificate window

10. Select **Signature Verification** and click **Continue**.

    Note: If the objects you are going to restore are signed by an object signing certificate that was issued by a private CA, you need to import the CA certificate before proceeding with the next steps.

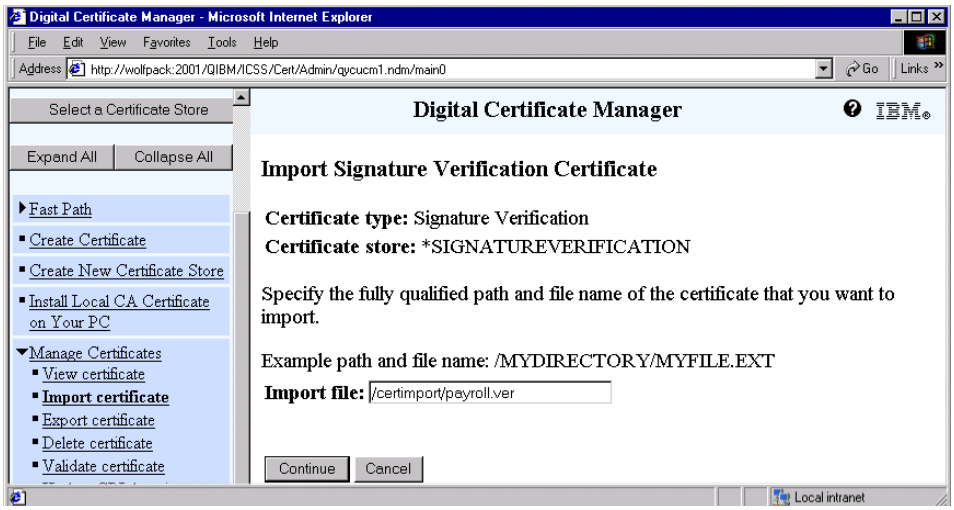


Figure 103. Import Signature Verification Certificate window

Enter the fully qualified file name of the file that holds the signature verification certificate you want to import. In this case it is the file that was FTPed by the payroll application administrator in 3.3.7, “Packaging the application” on page 155. This file is /certimport/payroll.ver.

11. Click **Continue**.

> **Note**
>
> If you do not specify the fully qualified or correct path and file name of the certificate that you want to import, you may be prompted for an encryption password for the file that contains the certificate rather than getting a message saying the file name was incorrect or not found. In this case, DCM is prompting you for the password before checking the filename you typed in on the previous window. Once the password has been entered, DCM then tries to do the import. At this stage, DCM checks that the file name is valid and the password is correct.

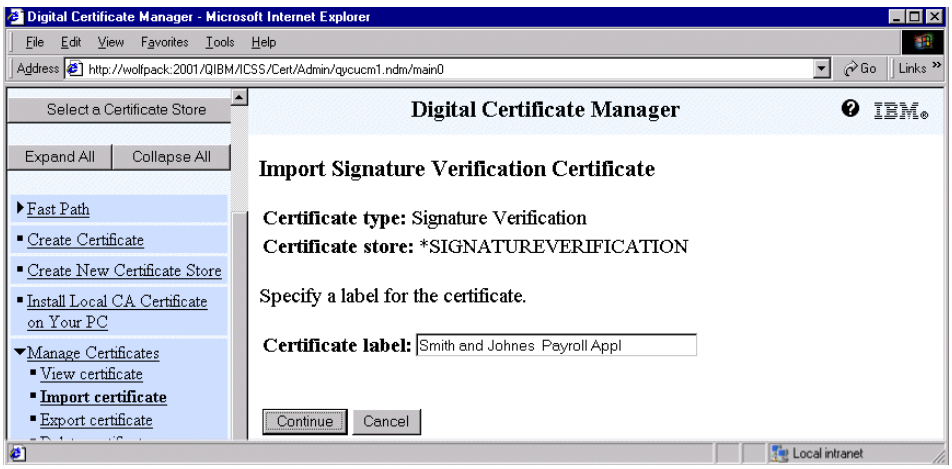


Figure 104. Import Signature Verification Certificate window: Label prompt

12. Specify a label for the new certificate and click **Continue**. Because this name uniquely identifies the certificate in the certificate store, you must specify a unique label. Use a label that describes what this certificate is used for.
13. You should then get a confirmation message saying that the certificate has been imported. Click **OK**.

Note

If the certificate store does not already contain a copy of the CA certificate for the CA that issued the signature verification certificate, you must import the CA certificate first. You may receive an error when you import the signature verification certificate if you do not import the CA certificate first.

These certificates can now be used to verify object signatures.

3.3.9 Restoring the application

The application objects can be supplied on media, such as tape or CD, or via save files. Both methods retain the signatures. Signature verification takes place automatically when objects are restored onto a system. The system value QVFYOBJRST (Verify object on restore) will determine the level of checking on the restored objects.

However, when restoring save files from media, the signatures of signed objects within a save file are not verified at this time. These signatures are verified when the objects are restored from the save file. The signature of the save file itself is checked before an object is restored from the save file. If the save file has a signature that is trusted, but is bad, no object will be restored from the save file. The QVFYOBJRST system values affects only *PGM, *SRVPGM, *MODULE, *SQLPKG, *STMF files with attached Java programs when restored from media or from a save file. Stream files without attached Java programs are also not affected by the system value, regardless of the value set.

The PAYRAPP save file has been transferred to the customer system and is stored in library PAYRCV. The customer's system administrator will restore the application by performing the following tasks:

1. Verify the signature of the PAYRAPP save file to ensure the integrity after the save file had been transferred to the customer system.
2. Restore the library PAYSLIP from the save file PAYRAPP.
3. Restore the payins1 and payfiles directories from the save file PAYIFSAPP in library PAYSLIP.

Since steps 2 and 3 are normal restore operations, they are not covered in this chapter. However, one example of restoring an object with a bad signature is shown in 3.3.11, "Restoring objects with bad signatures" on page 172.

3.3.10 Using DCM to verify object signatures

You can use DCM to verify the authenticity of digital signatures on objects. This will ensure that the data in the object has not been changed since the owner of the object signed it.

Before you can use DCM to verify signatures on objects, you must ensure that certain prerequisite conditions are met:

- The *SIGNATUREVERIFICATION store must be created to manage your signature verification certificates.
- The *SIGNATUREVERIFICATION certificate store must contain a copy of the certificate that signed the objects.
- The *SIGNATUREVERIFICATION certificate store must contain a copy of the CA certificate that issued the certificate that signed the objects.

To use DCM to verify signatures on an object or objects, complete the following steps:

1. Use a Web browser and enter the following URL to display the AS/400 Tasks page:

   `http://servername:2001)`

   When prompted, sign on with a user profile that has *AUDIT special authorities. To verify a signature on an object, the user profile does not need to have authority to the object itself.

   In this case, the customer's system administrator signs on with a user profile that just has *AUDIT special authorities.

2. Click **Digital Certificate Manager** from the AS/400 Tasks page.

   If you want to sign on to DCM with a user profile having *ALLOBJ and *SECADM special authorities, you would have to open the signature verification store first.

3. Click **Manage Signable Objects** on the navigation pane.

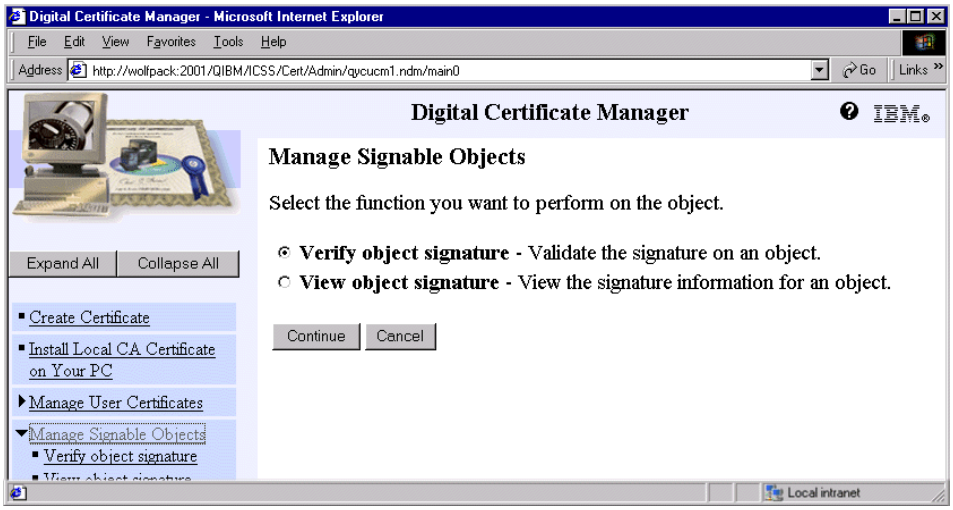


Figure 105. Manage Signable Objects window

4. Select **Verify object signature** and click **Continue**.

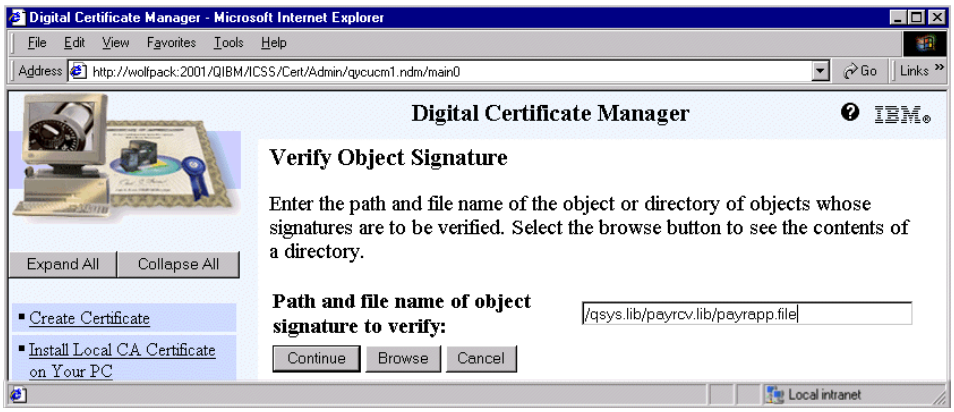


Figure 106. Verify Object Signature window

The system administrator is now checking the signature on the PAYRAPP save file that holds the entire payroll application. This step should be done to ensure the integrity of the entire package.

You can select the objects to be verified by specifying a fully qualified path and file name as well as using the browse or wildcard functions as described in 3.3.6, “Signing the application objects using DCM” on page 141.

5. Enter the path and file name of the object to be verified, which is /qsys.lib/payrcv.lib/payrapp.file.

   Note that the object name must start with a leading slash or you may encounter an error.

> **Note**
>
> To verify all objects in a specific directory you could, for example, enter `/mydirectory/*`. To verify all the programs in a specific library, you could enter `/QSYS.LIB/QGPL.LIB/*.PGM`. Wildcards can only be used in the last part of the path name. `/mydirectory*/filename` results in an error. If you use the Browse function to look at the list of library or directory contents, a wildcard should be entered as part of the path name before clicking Browse.

6. Click **Continue** to get the Processing Options for Verifying an Object Signature window.

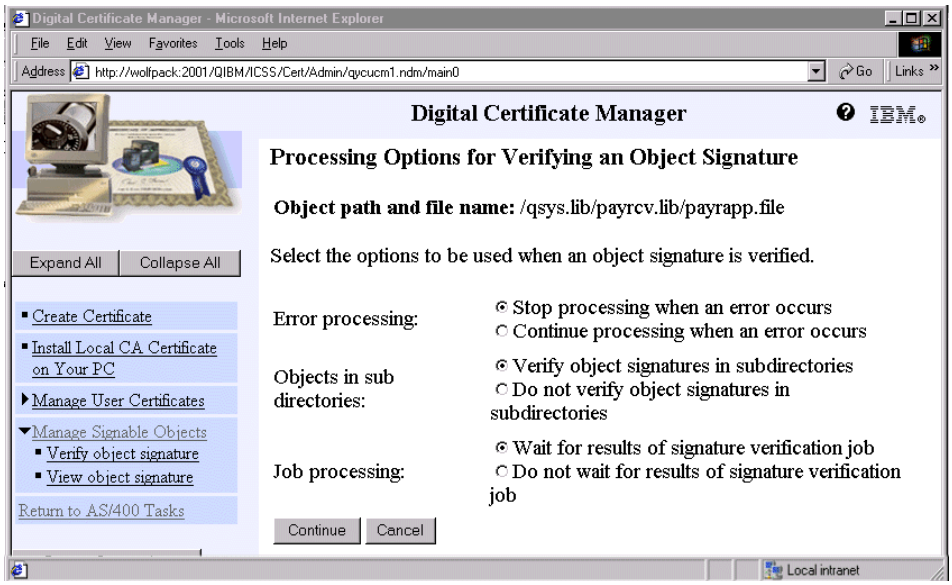


Figure 107. Processing Options for Verifying Object Signature window

Your choices in the window shown in Figure 107 are as follows:

Stop processing when an error occurs Select this to stop the signature verification when an error occurs. If verification for one object in the sequence fails, the verification process

stops without verifying signatures on any remaining objects.

Continue processing when an error occurs This option will ignore any errors and continue the signature verification process. The process attempts to verify all object in the job and the results are written to the file that you specified.

Verify object signatures in subdirectories Select this to perform signature verification for all signed objects in any subdirectories found in the path and file name that you specified.

Do not verify object signatures in subdirectories Select this to perform signature verification only on those objects in the main directory that you specified in the path and file name. The verification process ignores any subdirectories that are found under the specified path.

Wait for results of signature verification job Select this so that DCM waits until the signature verification process finishes for all objects before displaying verification results.

Do not wait for results of signature verification job DCM will run the signature verification job in batch mode and write the result of the job to a spool file. You can check the job results at a later time. Selecting this option is useful when you are verifying a large number of object signatures, since you can continue using DCM to perform other tasks while the verification process runs.

7. Since the administrator checks for only one object, the default settings are used. Click **Continue**.

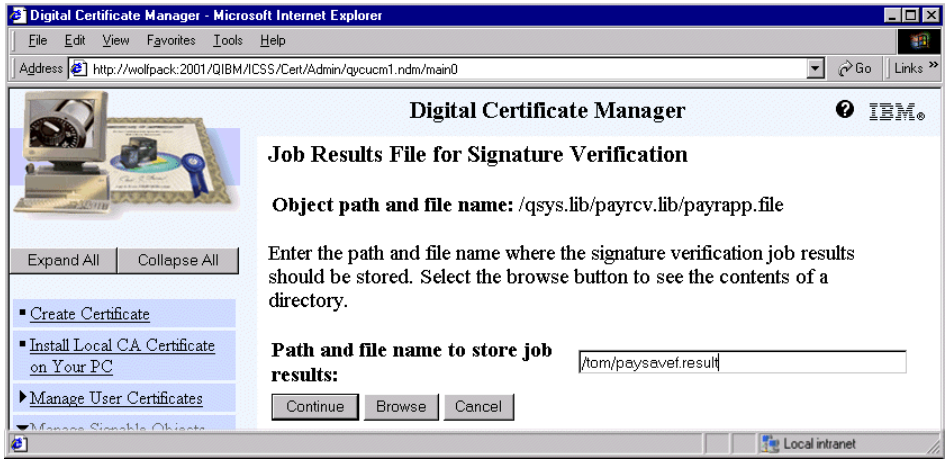


Figure 108. Job Results File for Signature Verification window

Results for the current job are appended to the end of an existing results file. Therefore, the file may contain results from any previous jobs as well as the current job.

8. Specify the fully qualified path and file name to use for storing job results for the signature verification operation, as follows:

   `/tom/paysavef.result`

   You can also enter a directory location and click **Browse** to view the contents of the directory to select a file for storing the job results.

9. Click **Continue**.

If you had selected **Wait for results of signature verification job** (the default), you will get the Verify Object Signature job output window, shown in Figure 109.

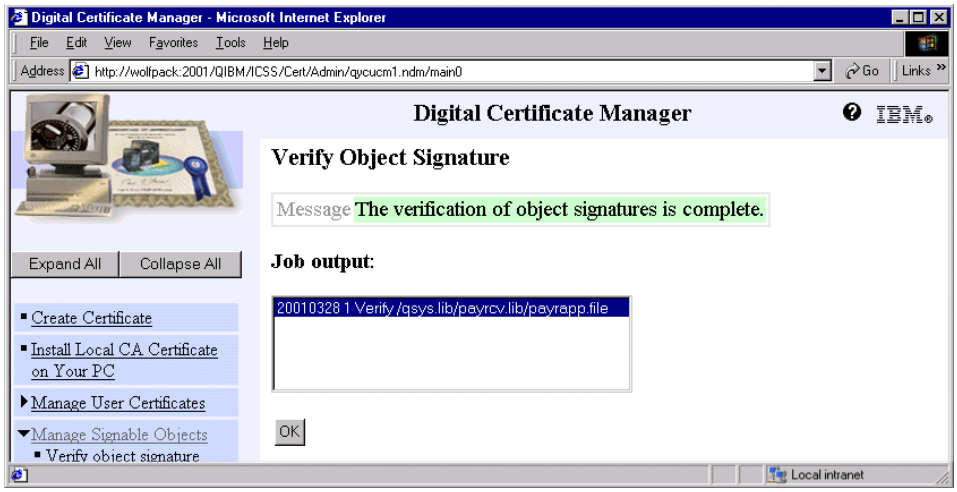


Figure 109. Verify Object Signature window: Job output

If you had selected **Do not wait for results of signature verification job**, you will get the following message:

```
The job was submitted to verify objects. To view the job results, see the
job QOBJSGNBAT in the job log.
```

The QOBJSGNBAT job is submitted by using the QDFTJOBD job description. By default, a batch job using this job description does not create a joblog spool file when the job completed normally. In all other cases you have a joblog that contains very good information about the errors that occurred.

In either case, the results will be posted to the results file that you specified, as shown in Figure 110. It is advisable to clear old results files or use new one.

```
Browse : /tom/paysavef.result
Record :       1   of       2 by  18                    Column :    1     89 by 131
Control :

...+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+....9.
************Beginning of data**************
             20010328        1 Verify               /qsys.lib/payrcv.lib/payrapp.file
************End of Data*******************
```

Figure 110. Results displayed in results file for verifying object

3.3.11 Restoring objects with bad signatures

In this section you see what happens when you try to restore an object with a bad signature. Let us assume the system administrator on the customer system received a second save file PAYRUPD which holds a program object PAYSECURE that solved an application problem. Since it is just one object, only the program object itself is signed, not the save file.

The system value QVFYOBJRST is still set to 3, meaning that all unsigned user-state objects as well as signed objects with valid signatures can be restored. Objects with invalid or bad signatures are not restored.

The administrator restores the object with the following command:

```
RSTOBJ OBJ(*ALL) SAVLIB(PAYSLIP) DEV(*SAVF) SAVF(PAYSECURE) RSTLIB(PAYSLIP)
```

With the current settings of the QVFYOBJRST system value the user gets the following error message:

```
                        Additional Message Information

Message ID . . . . . . :   CPD37A1       Severity . . . . . . . :   20
Message type . . . . . :   Diagnostic
Date sent  . . . . . . :   03/30/01      Time sent  . . . . . . :   23:26:37

Message . . . . :   PGM PAYSECURE in library PAYSLIP not restored.
Cause . . . . . :   Signature verification failed for PGM PAYSECURE in library
  PAYSLIP and the object could not be restored. An object with a signature
  verification failure cannot be restored when the Verify Object on Restore
  (QVFYOBJRST) system value is set to not allow restore of objects with a
  signature verification failure.
Recovery  . . . :   Do one of the following:
    -- Restore the object from a different saved version that contains the
  object with a valid signature.
    -- Restoring this version of this object may be a security risk but you
  can use the Change System Value (CHGSYSVAL) command to change the QVFYOBJRST
  system value to allow restore of objects with a signature verification
                                                                      More...
Press Enter to continue.
```

As the person who is in charge of restoring the object, you can still decide whether you want to restore the object anyway and take the risk of having an object that has been tampered with. If you do so, which means you set the system value QVFYOBJRST to ‘1’ and restore the object, you can check the object integrity afterwards by using the `CHKOBJITG` command, as follows:

```
CHKOBJITG OBJ('/qsys.lib/payslip.lib/paysecure.pgm')
   OUTFILE(QTEMP/PAYSECVFY)
```

Displaying the result file for an object with a bad signature shows the following information.

```
                    Display Physical File Member
File . . . . . . :   PAYSECVFY         Library  . . . . :   QTEMP
Member . . . . . :   PAYSECVFY         Record . . . . . :   1
Control  . . . . .                    Column . . . . . :   1
Find . . . . . . .
*...+....1....+.....//..+....5....+....6....+....7....+....8....+//.+....3
1040201092934AS4A 1   *PGM    QDFTOWN   BADSIG    0USENU   ¬a²H/Ç4  /qsys.lib/payslip.lib/paysecure.pgm
                      ****** END OF DATA ******
```

As you have seen in this example, you have various choices when restoring an object that has a signature violation. However, if in doubt always request the failing object again to ensure integrity.

3.3.12 Viewing an object signature

Objects can be signed with different certificates, which means an object can have multiple signatures. There are two ways of displaying signature information for a signed object:

- Using DCM to display the signatures, as described in this section
- Using the API QydoRetrieveDigitalSignatures, as described in 3.4.3, “Retrieving object signatures” on page 178

To display the signatures for the payroll application program PAYRMAIN perform the following steps:

1. Use a Web browser and enter the following URL to display the AS/400 Tasks page, af follows:

   `http://servername:2001)`

   When prompted sign on with a user profile that has at least Read and Execute data rights to the objects you want to display the signatures for.

2. Click **Digital Certificate Manager** from the AS/400 Tasks page.

   If you want to sign on to DCM with a user profile having *ALLOBJ and *SECADM special authorities, you would have to open the signature verification store first.

3. Click **Manage Signable Objects** on the navigation pane.

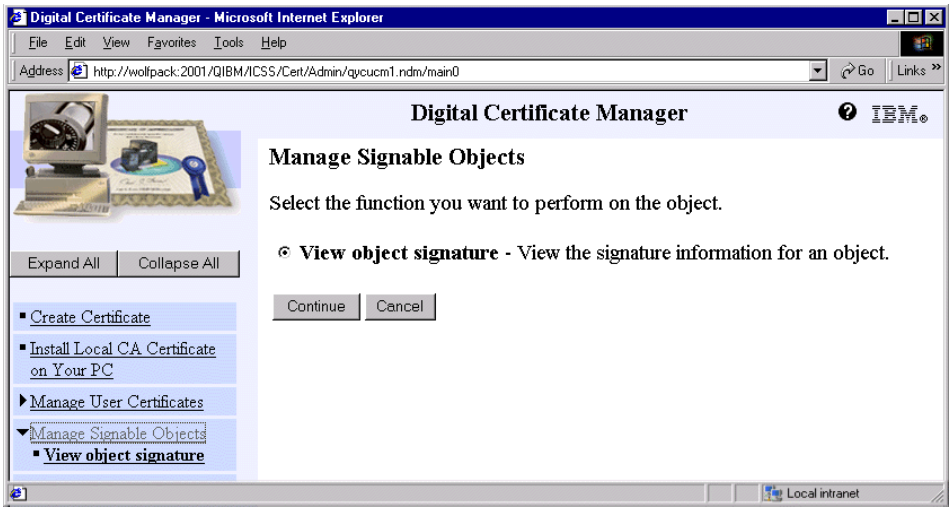


Figure 111. Manage Signable Objects window

Since the user profile that is signed on to DCM has no special authorities at all, only the View object signature option is displayed.

4. Select **View object signature** and click **Continue**.

   The View Object Signature window is displayed.

5. Enter the fully qualified path and file name of the object you want work with, beginning with a leading slash:

   `/qsys.lib/payslip.lib/payrmain.pgm`

6. Click **Continue**.

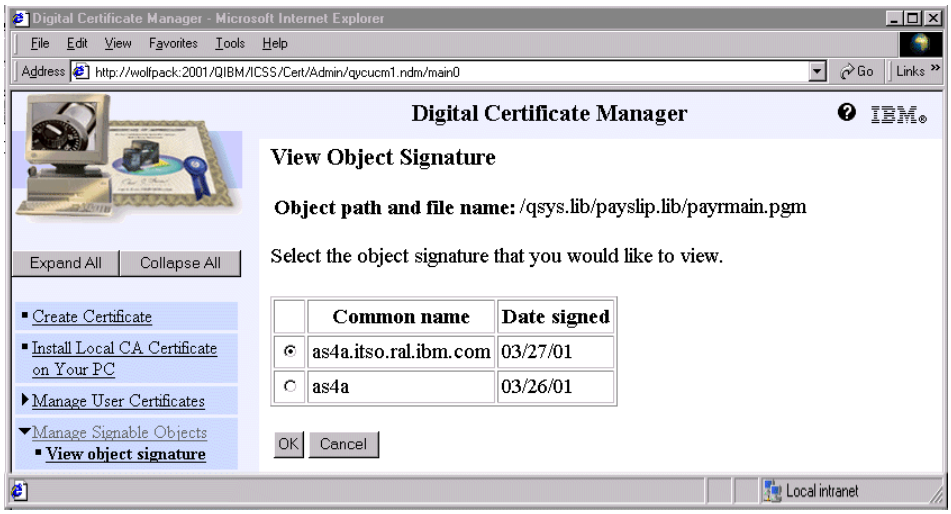


Figure 112. View Object Siganture window

The specified object `/qsys.lib/payslip.lib/payrmain.pgm` was signed by two different object signing certificates.

7. Select the object signature you want to view and click **OK**.

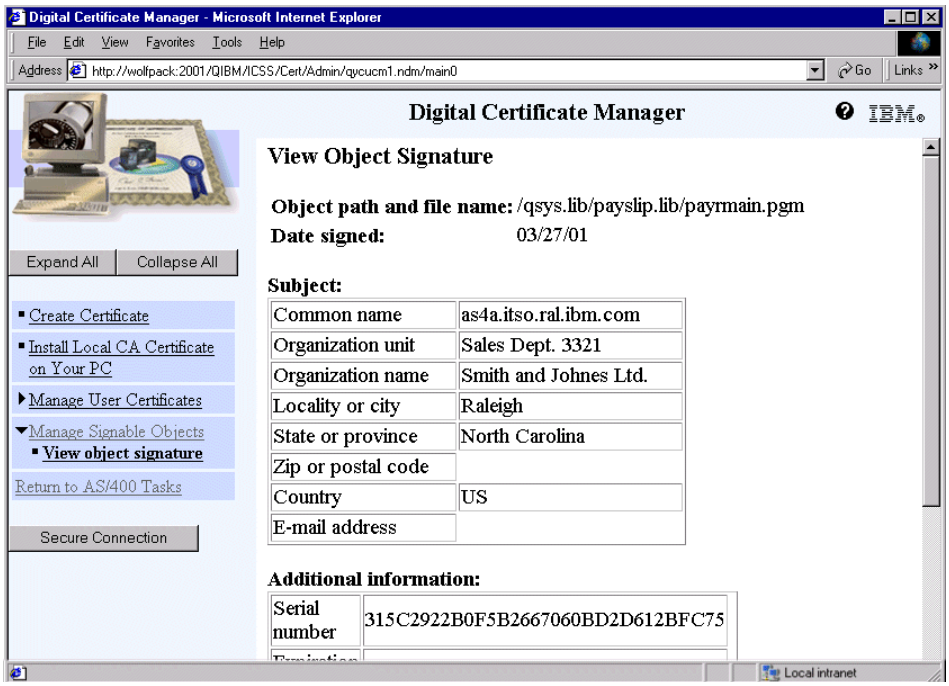


Figure 113. View Object Signature window - Signature details

3.4 Object signing application programming interfaces

OS/400 provides application programming interfaces (APIs) that allow customers and business partners to write their own applications for performing specific system tasks. These applications, for example, can be used to automate certain routine tasks. Also the object signing APIs could be used by a business partner to automate the object signing and signature verification process, so that no administrator would have to perform these tasks manually.

There are three application programming interfaces (APIs) related to object signing at V5R1:

- To sign an object or objects, you can use the OPM API QYDOSGNO, or the ILE API QydoSignObject.
- To verify the signature on an object you can use the OPM API QYDOVFYO, or the ILE API QydoVerifyObject.

- To retrieve information about the signature, or signatures, on an object you can use the OPM API QYDORTVO, or the ILE API QydoRetrieveDigitalSignatures.

We wrote three programs to demonstrate the object signing APIs. One signs objects, another displays details of the signatures on an object, the third verifies object signatures.

Note that the user profile that is using these APIs do not need to have any special authorities.

All program sources are available for download. See Appendix G, “Using the additional material” on page 471 to find how to obtain these program sources. For programming advice, see 6.1, “Programming techniques for using APIs and C functions in ILE RPG” on page 335. To find documentation detailing these APIs look in *Security APIs* found in the iSeries Information Center by clicking **Programming->CL and APIs->APIs by category->Security**.

3.4.1 Signing objects

The program SGNOBJ can add signatures to an object or a group of objects. It requires three parameters:

- The path name to the object, or objects, to sign
- The application identifier in the DCM to use
- The path name to the file in which the signing results will be written

SGNOBJ has been written to be called from a command or from another program.

We have also written a very simple command, `SGNOBJ`, to call the program SGNOBJ. For example:

```
SGNOBJ OBJECT('/qsys.lib/mylib.lib/savf.file') APPID(sales)
    LOG('sales/logs/sign.log')
```

SGNOBJ simply accepts the parameters, formats them, and calls the API QydoSignObject. If no errors occur, it sends a message stating the objects were signed. Otherwise, it relays the error message to its caller.

The APIs used in this program are:

- **QydoSignObject**: Digitally sign an object or objects
- **QMhSndPm**: Send a program message

If you want to understand this program in detail, please look at the program source.

3.4.2 Verifying object signatures

The program VFYSGN can verify the signature on an object or group of objects. It requires two parameters:

- The path name to the object, or objects, to verify
- The path name to the file in which the results will be written

VFYSGN has been written to be called from a command or from another program.

We have also written a very simple command, `VFYSGN`, to call the program VFYSGN. For example:

```
VFYSGN OBJECT('/qsys.lib/mylib.lib/savf.file')
    LOG('sales/logs/verify.log')
```

VFYSGN simply accepts the parameters, formats them, and calls the API QydoVerifyObject. If no errors occur, it sends a message stating the object, or objects, had valid signatures. Otherwise, it relays the error message to its caller.

Note

QydoVerifyObject returns an error if one or more objects had no valid signature. If an object has more than one signature, but at least one is valid, QydoVerifyObject does not treat this as an error. If this possibility is important to you, you should check the log file regardless of the status returned by the QydoVerifyObject API.

The APIs used by this program are:

- **QydoVerifyObject**: Verify object or objects signatures
- **QMhSndPm**: Send a program message

If you want to understand this program in detail, please look at the program source.

3.4.3 Retrieving object signatures

The program RTVSGN retrieves the details of the signature, or signatures, on an object. It requires one parameter, the path name to the object to retrieve the signature from.

RTVSGN has been written to be called from a command or from another program.

We have also written a very simple command, RTVSGN, to call the program RTVSGN. For example:

```
RTVSGN OBJECT('/qsys.lib/mylib.lib/savf.file')
```

RTVSGN simply accepts the parameter, formats it, and calls the API QydoRetrieveDigitalSignature. If no errors occur it sends a messages stating the common names of the certificates used sign the object. Otherwise it relays the error message to its caller.

The APIs used in this program are:

- **QydoRetrieveDigitalSignature**: Verify object or objects signatures
- **QMhSndPm**: Send a program message

If you want to understand this program in detail, please look at the program source.

3.5 Backup considerations

This section discusses the various issues you have to deal with when performing backup and recovery tasks for the object signing and signature verification environment.

3.5.1 Save and restore implications

In order to restore signed objects on a system that is verifying object signatures, the signature verification certificate must be present in the signature verification certificate store. You can import the certificate, or certificates, in the normal way. However you can also use save and restore functions.

Only the signature verification certificate store (VFYSGN.KDB) is saved by the Save Security Data (SAVSECDTA) or Save System (SAVSYS) command. The object signing certificate store (SGNOBJ.KDB) is not. You may need to consider this as part of your disaster-recovery planning or if moving from one system to another. The certificate stores are marked to prevent them from being saved during normal save operations.

The signature verification store is saved using SAVSECDTA or SAVSYS because the certificates in this store are needed to verify the signatures of objects when they are restored on the system, so this certificate store needs to be there before the other objects get restored. The object signing certificates are not needed for that, so the object signing certificate store is saved/restored with other IFS objects.

When the signature verification store is created, the “no save” attribute is set on so that it cannot be saved with a SAV command. This prevents the object from being saved/restored twice, which could result in the potential loss of data.

Important

The RSTUSRPRF *ALL command is needed to restore the signature verification certificate store. Be aware this replaces the old certificate store entirely. If the RSTUSRPRF command finds there is no signature verification certificate store on the save media, the local system’s certificate store is deleted.

Objects not restored

Two diagnostic messages are used when objects are not restored during a restore operation. They are sent to the joblog.

- CPC37A0 is posted when a system or inherit state object is not restored due to being unsigned and when the QVFYOBJRST does not allow restore of unsigned objects.
- CPD37A1 is posted when system or inherit state objects are not restored due to an invalid signature and QVFYOBJRST has been set to not allow the restore of objects with an invalid signature.

3.5.2 Copying a signed object

Users are allowed to copy a signed OS/400 object but some conditions apply to the copying of signatures.

Signatures copied

Signatures are copied with the rest of the object using the OS/400 command CRTDUPOBJ or SAVxxx and RSTxxx.

Signatures not copied

Signatures are not copied if a user tries to copy an object by FTPing a file or uses a cut-and-paste operation. The OS/400 object signature in both cases is dropped. This is because the target system may not run OS/400 and its arrival could cause problems on the target system.

3.6 Problem determination

As already mentioned in the previous chapter, the messages provided by DCM have been improved. You can see immediately if the operation was successful, since the message has a green background. Unsuccessful operations or errors have a red background.

When errors occur when setting up DCM for object signing or when you try to digitally sign an object or verify the signature on an object, always look in the joblogs of the HTTP ADMIN server jobs running in the QHTTPSVR subsystem.

Some of the error messages posted within DCM point to the exact cause of the problem; others are not so specific but suggest you go to the HTTP ADMIN server joblogs for further information. In the latter case the message shown in Figure 114 is displayed in DCM.

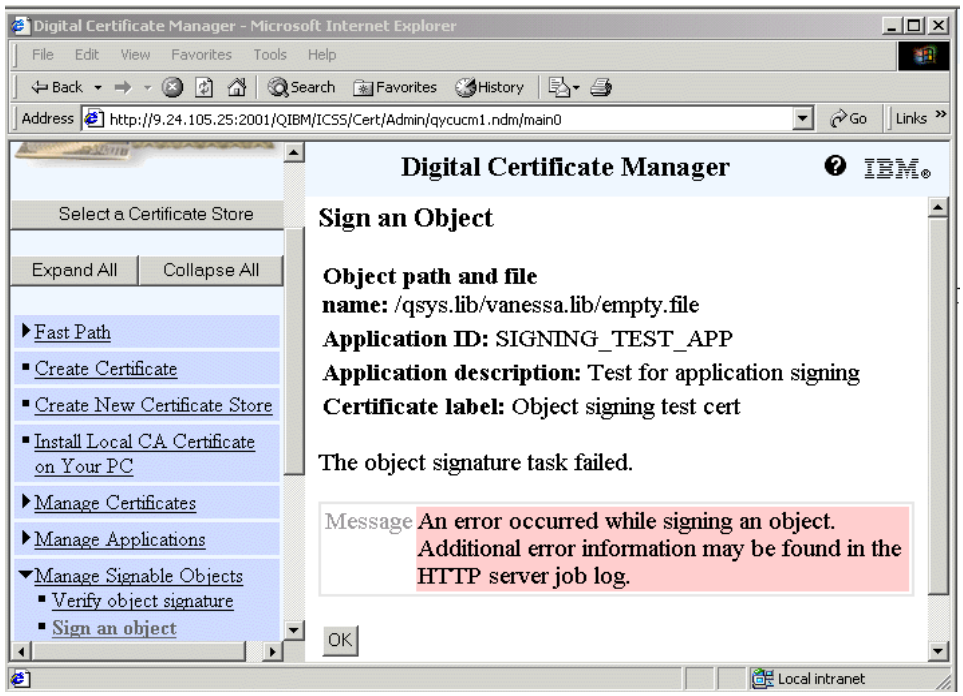


Figure 114. Error generated if application ID mapped to certificate has been deleted

Note that there are several HTTP ADMIN server jobs running under the QHTTPSVR subsystem. You may have to look into more than one joblog to find the error information you are looking for.

If you perform object signing or signature verification tasks in a batch job, examine the joblogs for more information. The job name you have to look for when using batch jobs initiated through DCM is QOBJSGNBAT.

3.6.1 Trace utilities

The Trace TCP/IP Application (`TRCTCPAPP`) command may be used to document problems related to digital signatures of AS/400 objects. The application *CERTSRV is used to start tracing for certificate functions. This trace facility should only be used when requested by IBM support personnel.

1. Start the trace by using one of the following commands:

   ```
   TRCTCPAPP APP(*CERTSRV) SET(*ON)
   ```

   This command starts traces for all system functions that are certificate related, such as DCM, VPN, Object Signing, or SSL. However, if you suspect the problem is in just the object signing area, start the trace with the following command:

   ```
   TRCTCPAPP APP(*CERTSRV) SET(*ON) CERTTYPE(*OBJSIGN)
   ```

2. Recreate the problem.
3. Stop the trace with the following command:

   ```
   TRCTCPAPP APP(*CERTSRV) SET(*OFF)
   ```

4. Lastly gather the output file data and other documentation required by IBM to analyze the problem. If you traced just for object signing `CERTTYPE(*OBJSIGN)` the following two spool files will be created in the current job:

   ```
   QPTOCSERVE  PRT01       YDO009441    HLD
   QPCSMPRT    PRT01                    RDY
   ```

   If you specified to trace for all certificate related services you also get a trace file containing DCM-related entries in the following directory:

   ```
   /QIBM/UserData/ICSS/CertSvcs/log
   ```

   In addition to the trace and spool files, you may also want to collect the files in:

   ```
   QIBM/UserData/ICSS/Cert/Signing
   ```

   Send the SGNOBJ.KDB file only if absolutely necessary, since this database also contains the private keys of the object signing certificates. However, the object signing database cannot be accessed without the certificate store password.

Job trace

When running a job trace of the failing operation the following points should be remembered:

- Use the new trace job alternative command (`STRTRC`)
- Perform the end trace and print trace commands in a batch environment

Normal problem determination (PD) processes should be followed for problems related to signed AS/400 objects. Data that may be helpful to assist resolve a problem include:

- A save of the objects being restored.
- A save of the certificate database from the QIBM subdirectory. This is found in QIBM/UserData/ICSS/Cert/Signing.
- Any relevant logs, including the LIC log.
- Any trace output.

3.6.2 Return codes

When performing object signing tasks in DCM several errors generate the error CPFB730: `Attempt to work with object signing certificates failed with unexpected return code xyz`. This is a very generalized message.

Some of the common return codes and their meaning are:

RC34 Validation error, the certificate is not valid.

RC105 No CRL entry. This error occurs when you use CRL checking for object signing certificates where the OS/400 LDAP client does not find a CRL at the defined location.

RC109 Error in get key by label. The application ID named a certificate that no longer exists. This error can occur when you tried to use an objects signing application to sign objects where the assigned certificate has been deleted.

3.7 Auditing

Audit journals can be used to record signature-related incidents. If you sign an object or remove signatures from an object, you can set up auditing so that the changes are reflected in a journal.

3.7.1 Setting up auditing for object signing

To audit changes to digitally signed objects you have to set up the auditing journal as described in the *iSeries Security Reference,* SC41-5302. The

summary of steps to set up the audit journal for logging object signing activities is as follows:

1. Create a journal receiver.

   Example: `CRTJRNRCV JRNRCV(JRNLIB/AUDJRN) THRESHOLD(100000)`

2. Create the audit journal.

   Example: `CRTJRN JRN(QSYS/QAUDJRN) JRNRCV(JRNLIB/AUDJRN) MNGRCV(*SYSTEM) DLTRCV(*NO)`

3. Set the QAUDCTL system value.

   Example: `CHGSYSVAL SYSVAL(QAUDCTL) VALUE('*OBJAUD')`

   The QAUDCTL system value must have at least *OBJAUD as one of the auditing controls. It is worth noting that none of the options of the QAUDLVL have anything to do with signed objects. So it makes no difference if this system value is or is not being used. At present there is no option in QAUDLVL to turn on or off the auditing of changes to objects with digital signatures.

4. When you turn on object auditing via the system value QAUDCTL, objects that have the auditing attribute set so that auditing occurs will be audited. You can select auditing for objects using the Change Object Auditing (`CHGOBJAUD`) command. Only those objects set for auditing will have object signing changes audited.

   Example: `CHGOBJAUD OBJ(PAYSLIP/*ALL) OBJTYPE(*ALL) OBJAUD(*CHANGE)`

   To see whether or not an object will be audited use the Display Object Description (`DSPOBJD`) command.

   Example: `DSPOBJD OBJ(PAYSLIP/PAYRMAIN) OBJTYPE(*PGM) DETAIL(*FULL)`

   You find the object auditing information on the second page of the object description as shown in the following figure.

```
                    Display Object Description - Full
                                                          Library 1 of 1
Object . . . . . . . :   PAYRMAIN       Attribute  . . . . . :   CLP
  Library  . . . . . :     PAYSLIP      Owner  . . . . . . . :   PAYROLL
Type . . . . . . . . :   *PGM           Primary group  . . . :   *NONE

Change/Usage information:
  Change date/time . . . . . . . . . :   03/28/01  17:47:12
  Usage data collected . . . . . . . :   YES
  Last used date . . . . . . . . . . :   03/27/01
  Days used count  . . . . . . . . . :   1
    Reset date . . . . . . . . . . . :
  Allow change by program  . . . . . :   YES
Auditing/Integrity information:
  Object auditing value  . . . . . . :   *CHANGE
 Digitally signed . . . . . . . . : YES

                                                                  More...
Press Enter to continue.

F3=Exit   F12=Cancel
```

3.7.2 Audit entry types

Signing objects, performing actions on signed objects, and restoring signed objects generate journal entries in the QAUDJRN. Displaying the QAUDJRN and searching for object signing related entries is not an easy undertaking. The journal entries that contain information related to object signing are:

AF Used for authority failures

DO Used for object delete operations. This entry is mentioned for completeness since it does not contain signature related entries, but might be useful for detecting signed objects that have been deleted.

OR Used for object restore information

ZC Used for object change operations, such as signing an object.

For all audit journal entry types, IBM has provided template outfiles you can use to better analyze the audit journal entries. The related audit journal template outfiles are:

QASYAFJE/J4 Used to log AF entries for authority failures.

QASYDOJE/J4 Used to log DO entries for object delete operations.

QASYORJE/J4 Used to log OR entries for restore operations.

QASYZCJE/J4 Used for ZC entries that are generated when an object is changed. This includes signing of an object.

The difference between outfiles ending with JE and J4 is the outfile format. For example, when performing a DSPJRN command with outfile format type *TYPE4 you have to use the corresponding template file ending with J4. For analyzing signature-related entries, the JE format is sufficient.

The following example shows how to use the template files and to determine the meaning of the entry for a signing operation:

1. Create the outfile you use for the journal entries by copying the template file to a file of your choice, for example:

```
CRTDUPOBJ OBJ(QASYZCJE) FROMLIB(QSYS) OBJTYPE(*FILE) TOLIB(BARLEN)
  NEWOBJ(ZC) DATA(*YES)
```

2. Display the audit journal into the new outfile by selecting only the ZC entries, for example:

```
DSPJRN JRN(QAUDJRN) FROMTIME(032801 183500) ENTTYP(ZC) OUTPUT(*OUTFILE)
  OUTFILFMT(*TYPE2) OUTFILE(BARLEN/ZCE)
```

3. Create a query of the created file and select the fields to display. In the case of a ZC entry type, you need at least the following fields to identify the object and action or access type:

   - Object Name
   - Library
   - Object Type
   - Access Type
   - Date and Time
   - User Profile

   The selected fields depend on what you want to achieve with the data. The query we created shows the following output.

```
Position to line  . . . . .
Line   ....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+..
       CODE  TYPE  ACCESS   OBJECT      LIBRARY     OBJECT    PROGRAM     USER
                   TYPE     NAME        NAME        TYPE      NAME        PROFILE
000001  T     ZC        7   *N          *N          *STMF     QP0ZPCP2    BARLEN
000002  T     ZC       67   PAYRMAIN    PAYSLIP     *PGM      QP0ZPCP2    BARLEN
000003  T     ZC        1   SIGNAUD     QSYS        *LIB      QCMD        BARLEN
000004  T     ZC        1   SIGNAUD     QSYS        *LIB      QCMD        BARLEN
```

The Access Type for the highlighted entry is 67 and means *Sign object*.

You can use the same technique to create outfiles for other journal entries and analyze the journal.

The important information you need to determine the signature related actions is as follows:

AF entry type
The field in the QASYAFJE/J4 that contains signature-related information is the Violation Type field. The most important value and its meaning is:

F The object did not restore because the signature was bad

OR entry type
The field in the QASYORJE/J4 that contains signature-related information is the Signature Status field. The possible values have the following meaning:

B Signature was not in OS/400 format

E Signature exists but is not verified

F Signature does not match object content

I Signature ignored

N Object is not signable

U Object is not signed

S There is at least one valid signature on the object

ZC entry type
The field in the QASYZCJE/J4 that contains signature-related information is the Access Type field. The most important values and their meaning are:

67 Sign object

68 Remove all signatures from an object

69 Clear a signed object

Always keep in mind that many system and object activities create journal entries in the audit journal. So you should only use this kind of journal when a high level of security is required.

For more information on auditing, see *iSeries Security Reference,* SC41-5302.

Chapter 4. Using hardware cryptography support for SSL/TLS

The Secure Socket Layer (SSL) V3 and Transport Layer Security (TLS) V1 protocols are services-independent protocols, which are nearby identical and are used to make communications over public networks more secure. The protocols ensure data origination, integrity, and confidentiality.

To begin the conversation the client sends a request (ClientHello) to the server and ask for the digital ID, the server certificate, and a listing of the ciphers the client supports. Throughout this handshake the client and the server together generate a unique communication key, called a session key. All communication between these two systems will be encrypted using that specific generated key. A part of this handshake involves cryptographic computations using public and private keys. These computations are very performance-intensive operations due to the used algorithms and key lengths, which are between 512 and 2048 bits depending on the certificates and Cryptographic Access Provider products used.

In V5R1, the 4758 PCI Cryptographic Coprocessor for iSeries can be used to improve the performance during the SSL session establishment by off-loading the performance-intensive tasks from the main processor.

But the 4758 PCI Cryptographic Coprocessor for iSeries is not just limited to speeding up the SSL handshake; it can also be used for other cryptographic tasks, such as:

- Generate random numbers
- Support financial Personal Identification Number (PIN) processing
- Generate and validate digital signatures (not to be confused with V5R1 object signing capabilities)
- Encrypt and decrypt data
- Protect keys
- Generate Message Authentication Codes (MAC)

The purpose of this chapter is to show you how the 4758 PCI Cryptographic Coprocessor for iSeries can be utilized for SSL/TLS-enabled applications. For more information about the 4758 Cryptographic Coprocessor card and its supported functions refer to *4758 PCI Cryptographic Coprocessor for iSeries* found in the iSeries Information Center by clicking **Security->4758 PCI Cryptographic Coprocessor for iSeries**.

4.1 Available cryptographic coprocessor adapters

The PCI 4758 Cryptographic Coprocessor for iSeries adds highly secure cryptographic processing capabilities to the iSeries server. Cryptographic coprocessor adapters are available for several AS/400 and iSeries models. Older versions of cryptographic coprocessor adapters, such as the SPD bus adapters #2620 or #2628 (withdrawn from marketing) cannot be used for SSL and have a limited support of cryptography functions compared to those of the newer PCI adapters.

The first adapter of the 4758 adapter family was supported with V4R4 on the AS/400 server. It was the Model 4758-001, which is also known under the marketing feature code #4800. The main benefit of the 4758 Cryptographic Coprocessor is that it provides the capability to store encryption keys in a tamper-resistant module. This module is located on the 4758 Cryptographic Coprocessor card. Inside the module, keys are stored in a battery backed-up memory. If, for example, an intruder tries to open or tamper with the module, special intrusion detection circuits are activated to prevent misuse of the card. This type of security for the cryptographic keys is important to customers in the banking, financial, insurance, and medical records industries. The 4758 Cryptographic Coprocessor meets the Federal Information Processing Standard (FIPS) PUB 140-1 requirements. Another benefit of the 4758 Cryptographic Coprocessor, which adds more security, is that it is a self-contained unit with its own operating system, called CP/Q++.

In contrary to older cryptography coprocessor adapters, the 4758 Cryptographic Coprocessor adapters are shipped with encryption capabilities disabled and therefore does not fall under the US export regulations. The cryptography functions are enabled by installing one of the Cryptographic Access Provider products (5722-ACx). In addition for the 4758 Cryptographic Coprocessor to work, the Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP - 5722-SS1 Option 35) product must be installed.

Feature 4800

Note that the 4758-001 coprocessor feature #4800 does not support SSL.

The latest model of the cryptographic coprocessor on the iSeries server is the 4758-023 coprocessor. There are two feature codes that can be used for ordering the cryptographic coprocessor. Feature code #4801 is used for those iSeries models where no service representative is needed to install the adapter (customer installation) and feature code #4802 is used for those models where a hardware service representative is required to install the

coprocessor. The feature code to be ordered depends on the system model as described in Table 4.

Only the 4758-023 coprocessor can be used with SSL-enabled applications. The new Web-based configuration interface introduced in OS/400 V5R1 allows you to easily configure the adapter using a configuration wizard. Certificates used during SSL handshake processing are managed through the Digital Certificate Manager (DCM).

The 4758-023 PCI Cryptographic Coprocessor supports all of the 4758-001 algorithms plus it adds support for triple-DES and provides improved SHA-1 and RSA performance.

4.1.1 Hardware requirements

The 4758-023 Coprocessor for iSeries can be ordered by specifying feature code #4801 or #4802. As previously mentioned in this chapter, feature code #4801 has to be ordered for systems where the adapter will be installed by the customer. For systems that require a hardware service representative to install the adapter, feature code #4802 must be ordered. Use Table 4 to determine what feature code to order.

Table 4. 4758-023 order feature codes and supported systems

Feature	Supported systems
4801	250, 270, 8xx, SB2, and SB3 Expansion towers 5074, 5075, 5079
4802	Expansion towers 5065 and 5066 when attached to AS/400 Models 6xx and 7xx

When ordering a new iSeries server including a 4758 PCI Cryptographic Coprocessor for iSeries, the coprocessor is shipped separately and has to be installed later.

Handling of the 4758 coprocessor

The 4758 coprocessor is a very sensitive device built to meet high security standards. Before unpacking and installing the adapter, read the installation and handling instructions carefully. Otherwise you may destroy the adapter.

When shipped, the 4758 PCI Cryptographic Coprocessor for iSeries is packed in an insulated box ensuring that the temperature of the adapter stays within a certain range.

If you allow the adapter to cool down below -15 degrees C (5 degrees F) the coprocessor destroys its factory setting and becomes unusable. If this happens, you have to contact your hardware service provider to order a new adapter.

Before inserting the adapter into your system you have to calculate the storage and performance requirements as described in *AS/400e server 270 and 8xx System Installation and Upgrade,* SY44-5966 to determine the PCI slot where the adapter can be installed. Depending on the iSeries model the maximum number of 4758 Cryptographic Coprocessor adapters in V5R1 is eight.

4.1.2 Software requirements

This section lists the software products that are required to use the 4758 PCI Cryptographic Coprocessor for iSeries for SSL processing.

- 5722-SS1: OS/400 V5R1M0

  Note that the 4758-023 coprocessor is also supported with OS/400 V4R5 with a limited function support.

- 5722-SS1 Option 35: Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP)
- One of the following IBM Cryptographic Access Provider licensed program products to enable the encryption capabilities of the 4758 coprocessor:
  - 5722-AC2: Cryptographic Access Provider 56-bit
  - 5722-AC3: Cryptographic Access Provider 128-bit
- 5722-SS1 Option 34: Digital Certificate Manager
- 5722-TC1: TCP/IP Connectivity Utilities
- 5722-DG1: IBM HTTP Server

Note

The United States Bureau of Export Administration classifies both Support Programs and the Coprocessors as "Retail Cryptographic Implementations". Thus, IBM can export these hardware and software products to essentially all customers (export restrictions remain in effect for a certain few countries and organizations).

For more information about the Cryptographic Access Provider products, refer to Chapter 7, “Ciphers and cryptographic product considerations” on page 373.

4.2 Planning considerations

We recommend that you do a thorough installation and configuration planning before setting up the 4758 PCI Cryptographic Coprocessor for iSeries. The time you invest prior to the configuration might save you time and money afterwards. For example, if you use DCM and request a server certificate from a well-known Certificate Authority (CA) where the private key is generated and stored in the 4758 Cryptographic Coprocessor adapter, you cannot install a second 4758 Cryptographic Coprocessor adapter to achieve load balancing with the same certificate afterwards. The reason is that the certificate’s private key is stored in the first adapter and cannot be transferred to the second adapter. In this case you would have to buy a new server certificate. This is just one example of what could happen without thorough planning. The following sections address some more issues you should consider when planning the configuration.

4.2.1 Planning for future growth

With OS/400 V5R1, the number of supported 4758 Cryptographic Coprocessors in a single iSeries server has been increased from three to eight adapters. The maximum number varies by iSeries model. In a typical environment you would start with one 4758 Cryptographic Coprocessor. As the number of SSL connection requests increase, you may want to add an additional 4758 Cryptographic Coprocessor and share the load between the first and second adapter. At this time you will find out whether you made the right choice in the first place. If you decided to store the server certificate’s private key in the 4758 Cryptographic Coprocessor, you have to request (buy) a new certificate. If you decided to store the certificate’s private key in a key file encrypted by the master key of the 4758 Cryptographic Coprocessor, you just need to update the device assignment for the certificate after the new

adapter has been properly set up. Of course, there might be security reasons to have the private keys stored in the hardware adapter. In these cases you cannot use multiple 4758 Cryptographic Coprocessor adapters for load balancing. Refer to 4.7, “Load sharing” on page 224, for more information on load balancing with the 4758 PCI Cryptographic Coprocessor for iSeries.

4.2.2 Security considerations

The 4758 PCI Cryptographic Coprocessor for iSeries has access controls that do not relate to the OS/400 access controls, such as user profiles or object authorities. This allows you to assign the roles and responsibilities to different people. For example, the security officer who manages OS/400 user profiles can be different from the security administrator who manages the security on the 4758 Cryptographic Coprocessor.

Among the most important information about the 4758 Cryptographic Coprocessor is the master key. This key is used to protect all information that is stored on the coprocessor or the keys that are stored outside the coprocessor in key files. During configuration using the configuration wizard you can specify whether the master key is made of one or three parts, which again allows you to split responsibilities among several people. The master key can be entered manually or automatically generated by the coprocessor. To take advantage of the automatic key generation, you should consider a second 4758 Cryptographic Coprocessor to which you can clone the master key. If there is no additional adapter, the master key cannot be retrieved and in case of a hardware error the existing public key algorithm (PKA) and Data Encryption Standard (DES) keys that were protected by the master key cannot be used anymore. More information about the configuration options when using the wizard may be found in 4.3, “Configuring the 4758 Cryptographic Coprocessor” on page 195.

The 4758 PCI Cryptographic Coprocessor for iSeries is a great adapter that enhances security and improves performance, but as you can imagine from the given information, if you do not do proper setup planning you might end up with a coprocessor that needs to be re-initialized in order to work again. This results in additional costs and time needed to rebuild the configuration and environment.

For a detailed description about 4758 Cryptographic Coprocessor security features and setup, refer to *4758 PCI Cryptographic Coprocessor for iSeries* found in the iSeries Information Center by clicking **Security->4758 PCI Cryptographic Coprocessor for iSeries**.

4.3 Configuring the 4758 Cryptographic Coprocessor

There are two different ways of configuring the 4758 PCI Cryptographic Coprocessor for iSeries. Prior to V5R1, you could configure the 4758 Cryptographic Coprocessor by using APIs in applications you wrote, or using example programs that were provided by IBM. The example programs are written in C and RPG and are available in *4758 PCI Cryptographic Coprocessor for iSeries* found in the iSeries Information Center by clicking **Security->4758 PCI Cryptographic Coprocessor for iSeries**. In V5R1, you can still use this method for configuring the 4758 Cryptographic Coprocessor, but the quickest and easiest way to configure the coprocessor is with the new Web-based configuration utility, as described in the following steps:

1. Launch the AS/400 Tasks page by using a Web browser and enter the following URL:

   `http://servername:2001`

   Where *servername* represents your iSeries host name. The options available from the AS/400 Tasks page vary depending on the installed program products.

   Make sure that the user profile that is used to perform the configuration of the 4758 Cryptographic Coprocessor has *SECADM and *IOSYSCFG special authorities.

2. Click **4758 Cryptographic Coprocessor** on the AS/400 Tasks page.

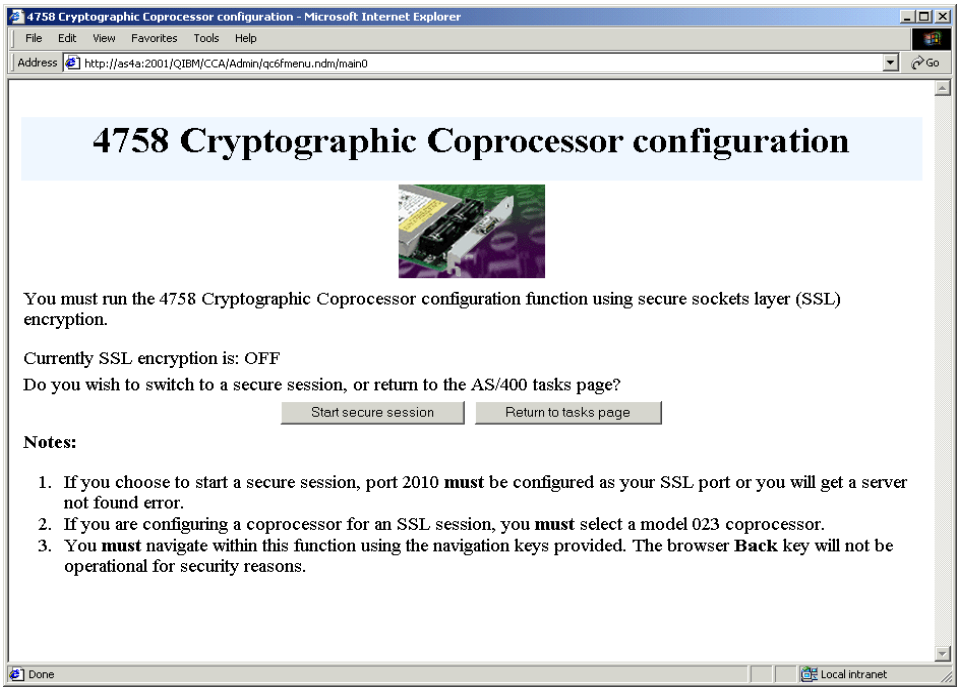


Figure 115. Starting the 4758 Cryptographic Coprocessor configuration window

The configuration of the 4758 can only be performed using a secured connection. The information shown in Figure 115 indicates that there is no secure session between the workstation you are working with and the iSeries Server you are connected to.

If SSL is not configured for the ADMIN server instance, you first have to configure the ADMIN server for SSL and assign a server certificate using DCM. If you are not familiar with that step refer to Appendix C, “Enabling SSL for the ADMIN server instance” on page 415.

3. Click **Start secure session** to restart the connection to the requested iSeries server as a secured session. When using a server certificate for the ADMIN server instance that was not issued by a well-known CA, you may encounter a warning message issued by the browser, such as a `security alter` message. Follow the directions given by the browser to accept the certificate and continue. If you receive a message that SSL has not been activated, refer to Appendix C, “Enabling SSL for the ADMIN server instance” on page 415, for information on how to enable SSL for the ADMIN server instance.

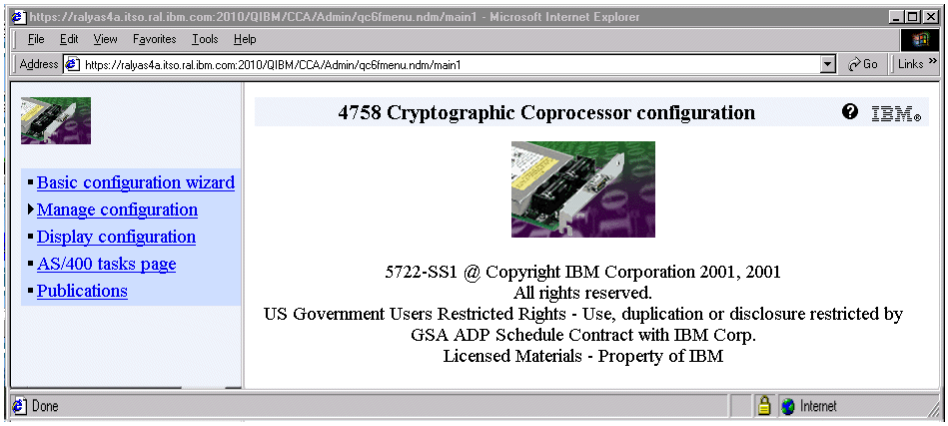


Figure 116. 4758 Cryptographic Coprocessor configuration window

4. Click **Basic configuration wizard** on the navigation pane. The wizard performs all steps that are required to configure the 4758 PCI Cryptographic Coprocessor for iSeries for SSL use.

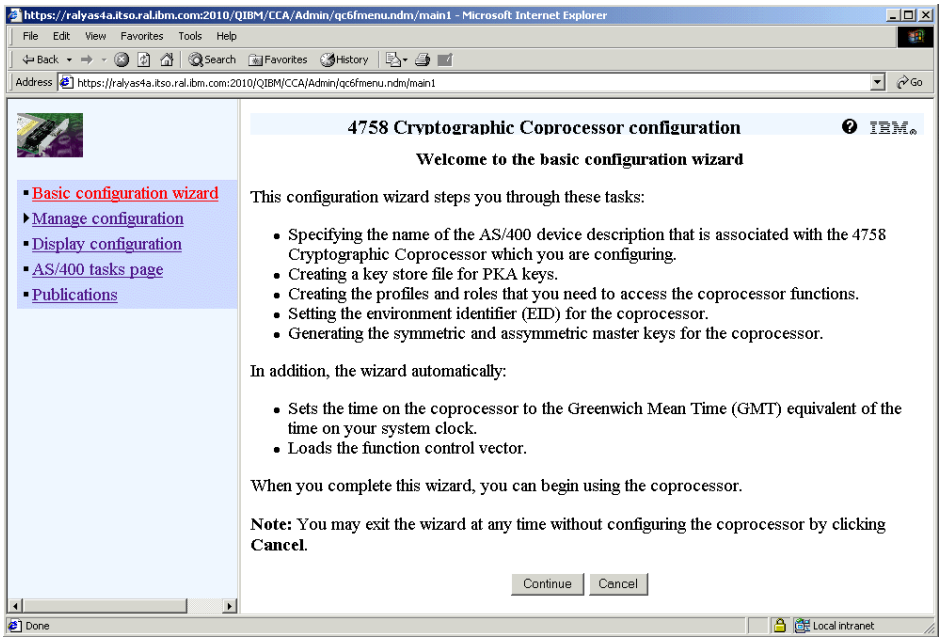


Figure 117. Welcome to the basic configuration wizard window

The welcome window is the first configuration window and it explains all steps the wizard will guide you through.

5. Click **Continue**.

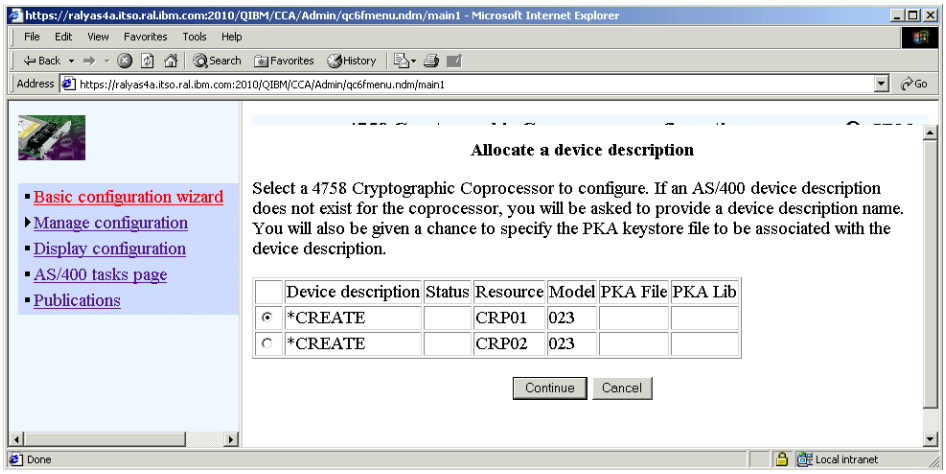


Figure 118. Allocate a device description window

6. Select the resource you want to create a new or an additional device for. In this case, the 4758 with the hardware resource name CRP01 is selected.
7. Click **Continue**.

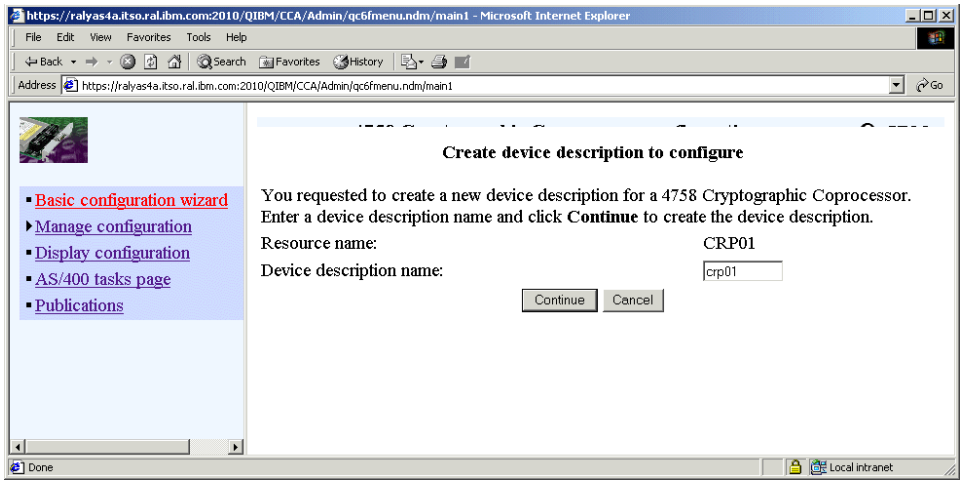


Figure 119. Create device description to configure window

Enter `CRP01` as the device description name.

For simplicity and easier management, it is a good idea to name the device after the hardware resource name, in this case CRP01. If user-written applications want to use a device that is named something other than the hardware resource name, they need to use the Cryptographic Resource Allocation (CSUACRA) API. The hardware resource name (for example, CRP01) is the default device name. If an application never calls CSUACRA, CCA looks for the device description with the name of the hardware resource. If an application calls CSUACRA, the CCA uses the device named on the call for the rest of the job (or until CSUACRD is called). The device description is used by CCA CSP to help direct cryptographic requests to the 4758 Cryptographic Coprocessor. Additionally, the device description gives your 4758 Cryptographic Coprocessor a default location for key store file storage.

8. Click **Continue**.

   The device description will automatically be created. Next you receive information messages stating that the device is being varied on and that it takes about one minute to become active.

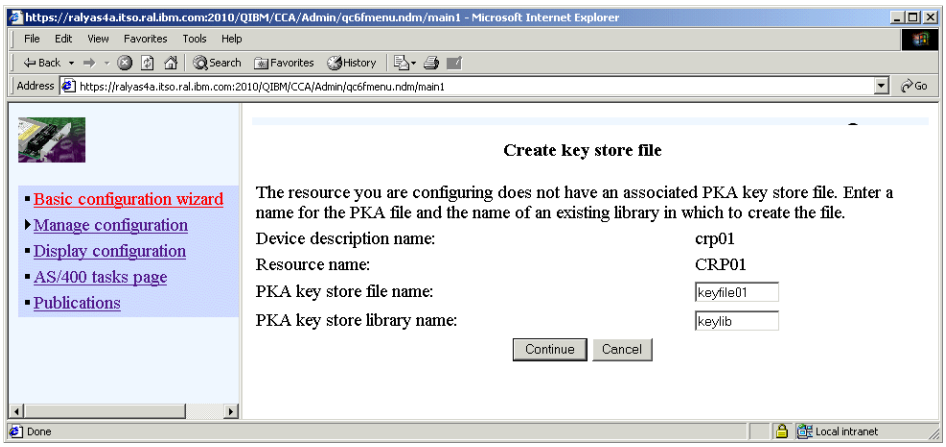


Figure 120. Create key store file window

   The Create key store file window requests information to be used to create the PKA key store file. PKA key store files are used to store private keys, which are encrypted by the master key of the 4758 Cryptographic Coprocessor. Make sure the library already exists before you click Continue.

9. Enter a name for the PKA file.

This can be a name for a new file or in case you configure a second 4758 Cryptographic Coprocessor that will be used for load balancing, a name of an existing one.

10. Enter a library that holds the PKA key store file. The library must already exist.
11. Click **Continue**.

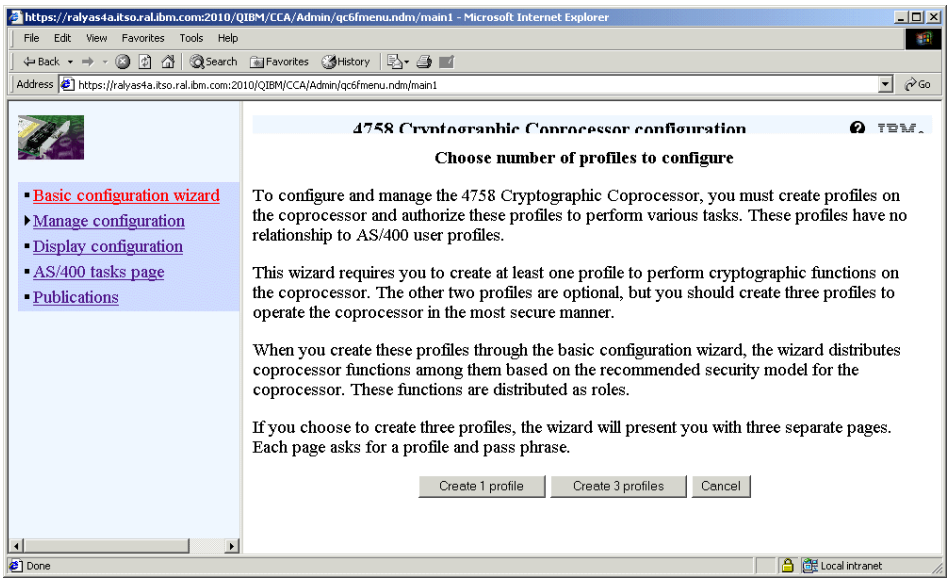


Figure 121. Choose number of profiles to configure window

The explanation shown in Figure 121 introduces the basic concepts of using profiles to manage tasks on the 4758 Cryptographic Coprocessor. This is one of the configuration steps that require that you already know how you would like to manage the 4758 Cryptographic Coprocessor environment as described in 4.2, “Planning considerations” on page 193. Generating three profiles, which is the preferred way, allows you to split responsibilities. However, in case you are the only IT person in the company you may want to operate the 4758 Cryptographic Coprocessor using only one profile. The functions an individual profile can perform are defined by roles. Appendix A, “4758 cryptographic coprocessor hardware commands” on page 403, contains the commands each profile can perform when initially created. Depending on your security needs you should consider creating more profiles and assign customized roles to them. For more information about the adapter security, profiles, and roles refer to *4758 PCI Cryptographic Coprocessor for iSeries* found in the

iSeries Information Center by clicking **Security->4758 PCI Cryptographic Coprocessor for iSeries**.

12. Click **Create 3 profiles**.

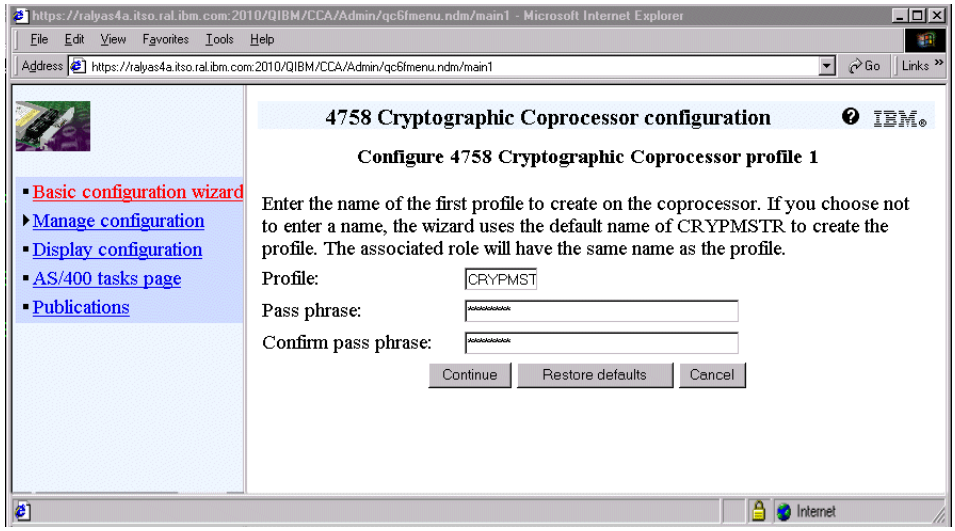


Figure 122. Configure 4758 Cryptographic Coprocessor profile 1 window

Based on the previous selection the wizard will prompt for a pass phrase for one or three profiles. The profile names are automatically entered by the wizard. As shown in Figure 122, the wizard starts with the CRYPMSTR profile, which is the only profile when managing the 4758 Cryptographic Coprocessor with one profile. In case of three profiles, you also have to enter pass phrases for the CRYPSEC and CRYPADMN profiles.

As the name *pass phrase* implies, do not use just a single six-character password. Entire sentences, such as ‘My grandmother likes to tango’ or ‘Wolfsheim ist ein schoenes Oertchen’ are much more difficult to guess than your dog’s name.

However, make sure you store the pass phrase in a secure place. There is no way to recover or override it. In case of a forgotten or lost pass phrase you have to re-initialize the 4758 Cryptographic Coprocessor and start from scratch - all your keys are lost. You may also consider creating a second master profile later on with a pass phrase that will be deposited at the bank.

13. Enter a pass phrase and click **Continue**.

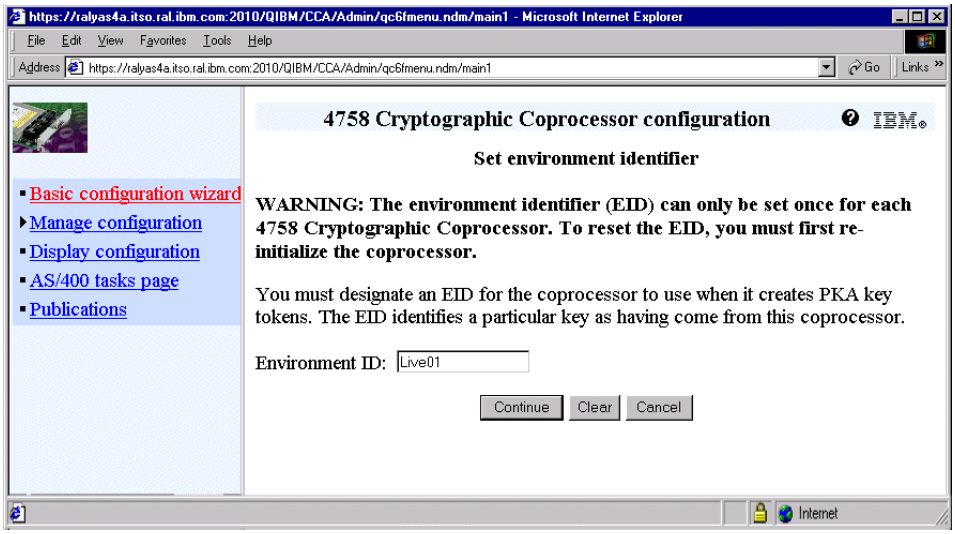


Figure 123. Set environment identifier window

This window is used to set the unique environment identifier (EID) for the 4758 Cryptographic Coprocessor. The EID operates like an iSeries system name, which identifies an iSeries server in a network. When you have multiple coprocessors in a system, the EID identifies them so that they can exchange PKA keys and clone master keys.

The EID is set just once on a 4758 Cryptographic Coprocessor, but you can create more than one device description per physical adapter.

14. Enter an EID and click **Continue**.

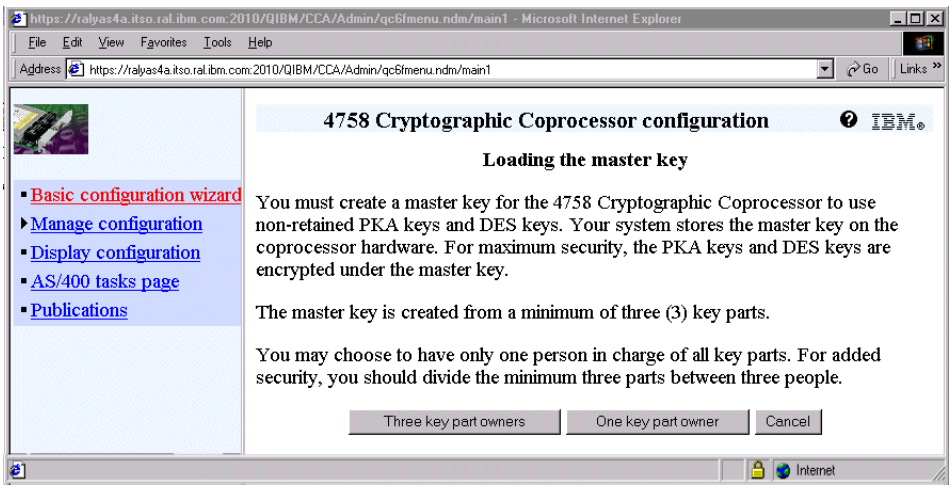


Figure 124. Loading the master key window

The next task the configuration wizard performs is setting the master key. The master key is stored on the 4758 Cryptographic Coprocessor adapter and is used to encrypt PKA and DES keys that are not stored on the coprocessor hardware (non-retained). As with having the choice of creating one or three profiles, you can also specify whether you want to have one person taking care of the three master key parts or divide the three master key parts between three different people. In this case we have decided to have one person who is in charge of the master key.

15. Click **One key part owner**.

    Remember, for better security you may choose to divide the master key parts between three people.

    Depending on the selected option the next window shown looks different. In this configuration where we have selected that one person owns the key, a window appears with three empty fields, each for one part of the master key. You can enter a master key of your own choice or have the system create a random key.

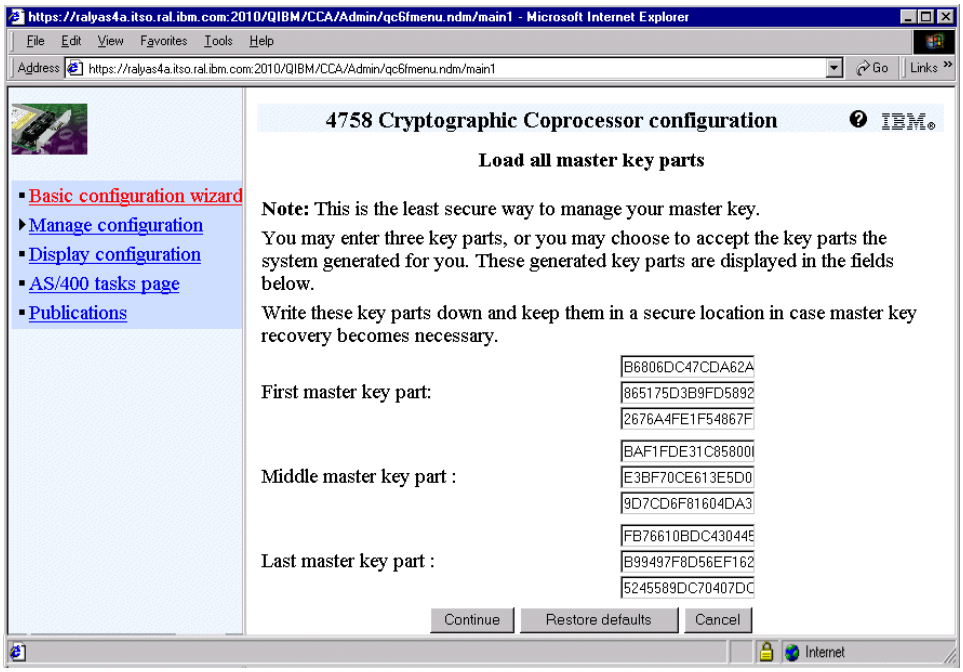


Figure 125. Load all master key parts window

The Load all master key parts window shows all parts of the master key. Write it down and store it in a very secure place. Depending on the browser settings all characters of key might not be shown on the window. Thus we do not recommend that you print the window. Printing the key may also compromise security if the output is printed in another room that many people have access to. Remember, the 4758 Cryptographic Coprocessor is a security device and therefore should be treated as such.

16. Click **Continue** after you have written the information down.

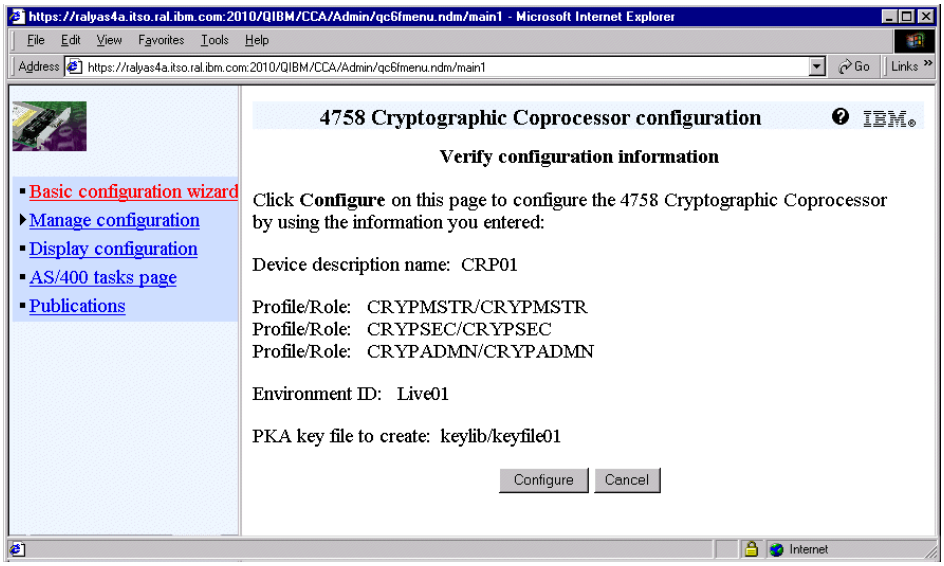


Figure 126. Verify configuration information window

The window in Figure 126 displays the chosen configuration options, such as the environment ID, profiles, and PKA key store file. By clicking the Configure button, the displayed configuration will be generated. Using the Cancel button aborts the creation. In that case the device has already been generated and has to be deleted. The OS/400 command `WRKCFGSTS CFGTYPE(*DEV) CFGD(*CRP)` displays the cryptographic devices and allows you to vary off and delete them manually.

17. Click **Configure**. The configuration wizard creates the profiles and sets configuration values. A message appears informing that the wizard has completed and that the completion results can be reviewed.

> **Note**
>
> At the time this redbook was written, we experienced problems when performing the Configure option. The Environment ID (EID) was not set. To solve this problem you can set the EID manually by clicking **Manage configuration->Attributes**, then select the new coprocessor device desciption, sign on with CRYPMSTR (1 profile) or CRYPADMN (3 profiles) and set the EID.

18. Click **OK** to close the information message window.

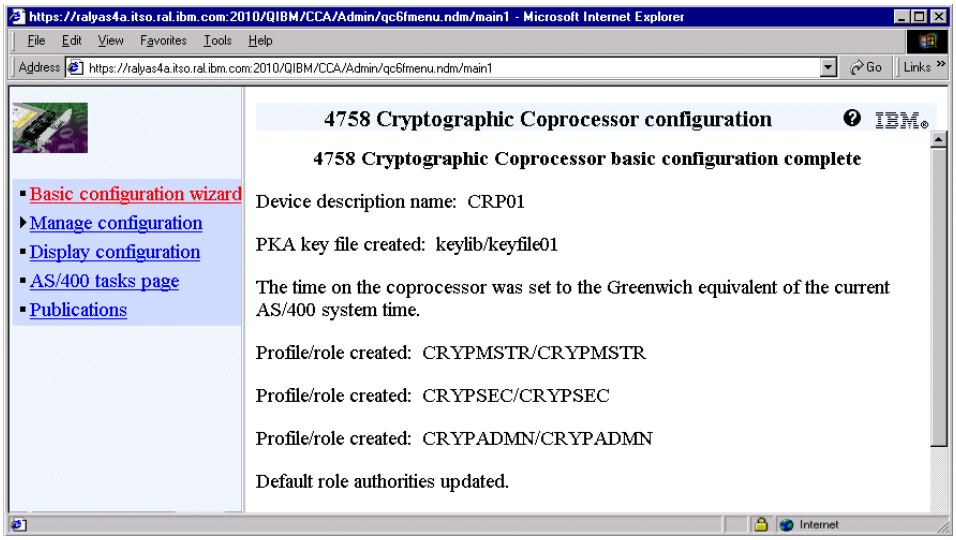


Figure 127. 4758 Cryptographic Coprocessor basic configuration complete window

4.4 Configuring DCM to utilize hardware cryptography for SSL

The Digital Certificate Manager (DCM) is the central management tool for certificate-related tasks, such as managing certificates, assigning certificates to SSL applications, using certificates for object signing, and so forth. Also the use of the 4758 Cryptographic Coprocessor for improving performance during SSL handshakes requires the DCM to perform the necessary configuration.

4.4.1 What you should consider first

There are certain considerations to be taken into account before configuring DCM to use the 4758 Cryptographic Coprocessor. First, you have to understand why configuration changes are required to enable the coprocessor to use certificates. In the next step you have to consider the implications of future growth and the restrictions. In this section we focus on the DCM management tasks as they relate to SSL. The standard certificate store that holds certificates used for SSL is the *SYSTEM certificate store.

DCM allows you to request certificates that are signed by a local Certificate Authority (CA), a public or well-known CA, or from a Public Key Infrastructure X.509 (PKIX) CA. When creating the certificate request in the *SYSTEM certificate store, DCM creates the certificate's public key with the

corresponding private key. The private key, which is kept secure, can be stored in three different ways:

- **Certificate store**: When choosing this method, the private key is stored in the current certificate store, in this case the *SYSTEM store. The 4758 Cryptographic Coprocessor has no access to the certificate's private key and therefore cannot use hardware cryptography for SSL.
- **Hardware**: Using this approach stores the private key on the 4758 Cryptographic Coprocessor. This option provides the highest security for storing the private key because of the tamper-respondent design of the coprocessor. However, only the coprocessor adapter that contains the certificate's private key can be used for SSL with this particular certificate.
- **Hardware encrypted**: Selecting this method uses the 4758 Cryptographic Coprocessor's master key to encrypt the certificate's private key and stores it in a keystore file. This option allows you to share a single certificate between different coprocessor's to support SSL handshake processing.

Depending on your company's security needs and policies you may want to store private keys as secure as possible. In this case you select the **Hardware** option when creating the certificate store. However, this option prevents you from later expansion to use multiple 4758 Cryptographic Coprocessors for load-balancing purposes. So if you are in doubt and there are no security constraints, you should select the **Hardware encrypted** option to create and store the private keys. Another aspect you may want to consider is whether you plan to export certificates including the private key. If this is a requirement you have to select the **Certificate store** option, but this one does not allow you to use the certificate by the 4758 Cryptographic Coprocessor.

As you can see there are some trade-offs. Therefore it is very important that you take a closer look into your company's organization, security needs, and future plans before starting to configure DCM. In fact, you need to decide at certificate request creation time which method you want to use.

4.4.2 Configuring DCM to use the 4758 Cryptographic Coprocessor

The information in this section guides you through the various DCM configuration steps that enable the 4758 Cryptographic Coprocessor to perform parts of the SSL handshake. If you already have a certificate assigned to a 4758 Cryptographic Coprocessor and you want to use the same certificate with another coprocessor, refer to 4.6, "Updating the 4758 Cryptographic Coprocessor device assignment" on page 218.

1. Start the AS/400 Tasks page using a Web browser and enter the following URL:

   `http://servername:2001`

   Where *servername* represents your iSeries host name. The options available from the AS/400 Tasks page vary depending on the installed program products.

2. On the AS/400 Tasks page click **Digital Certificate Manager** to start DCM. Refer to Chapter 2, “Digital Certificate Manager” on page 11, for more information about using DCM.

   The DCM window appears. It is separated into two parts: the major pane on the right and the navigation pane on the left, which depending on the selected store and user authority, shows the available options for the user signed on to DCM.

3. From the navigation pane, click **Select a certificate store**.

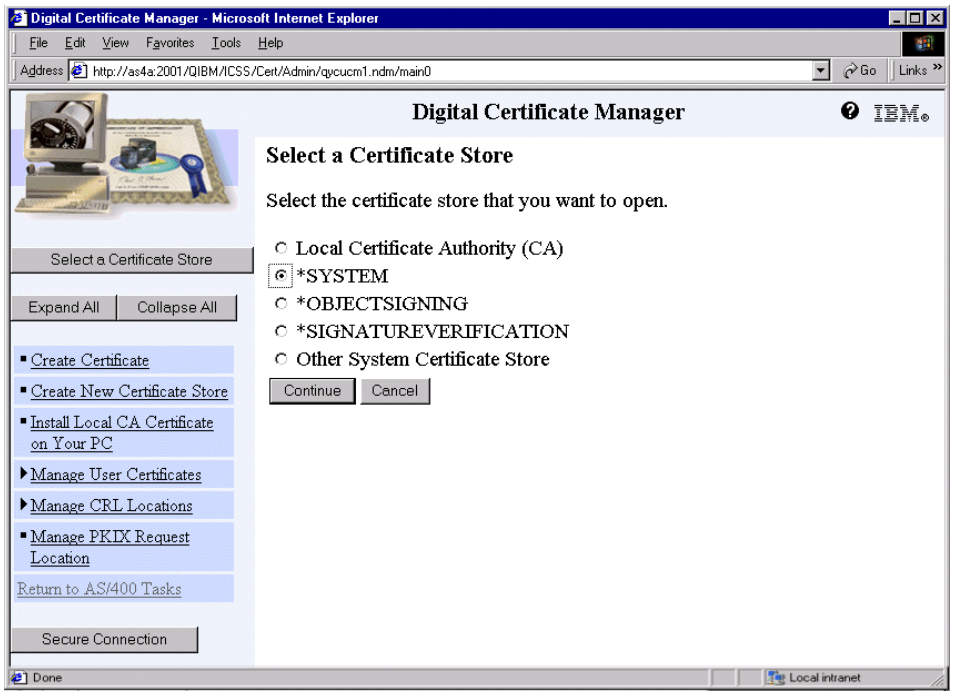


Figure 128. Select a Certificate Store window

4. Select the ***SYSTEM** certificate store and click **Continue**.

   The certificate store password prompt appears. The store is protected by a password that has been given by the administrator during the installation

process. The window also indicates what type of certificate store has been chosen, and where the certificate key database is stored, including the path and the file name.

5. Enter the password and click **Continue**. When using the Enter key instead of clicking Continue, you receive a NET.DATA error message.

   The navigation pane refreshes and shows all available options for the signed-on user in the *SYSTEM certificate store.

   During the next steps, you create a certificate request.

6. On the navigation pane, click **Create Certificate**.
7. Select **Server or client certificate** and click **Continue**.

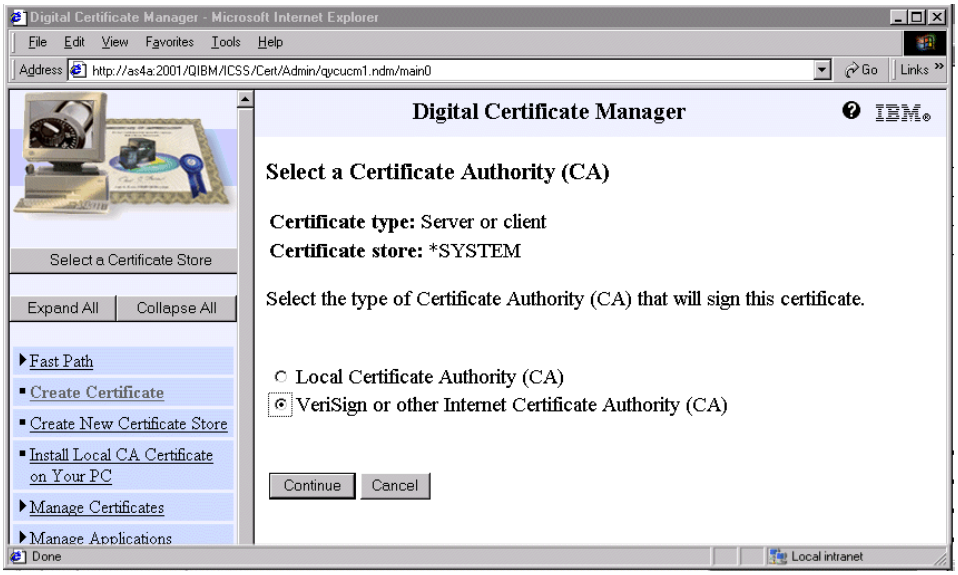


Figure 129. Select a Certificate Authority (CA) window

We have chosen to request a certificate from a well-known CA. The remaining steps as far as the 4758 Cryptographic Coprocessor is concerned are the same whether you request a certificate from a well-known CA or a local CA.

8. Select **VeriSign or other Internet Certificate Authority (CA)**.
9. Click **Continue** to proceed to the key storage selection window.

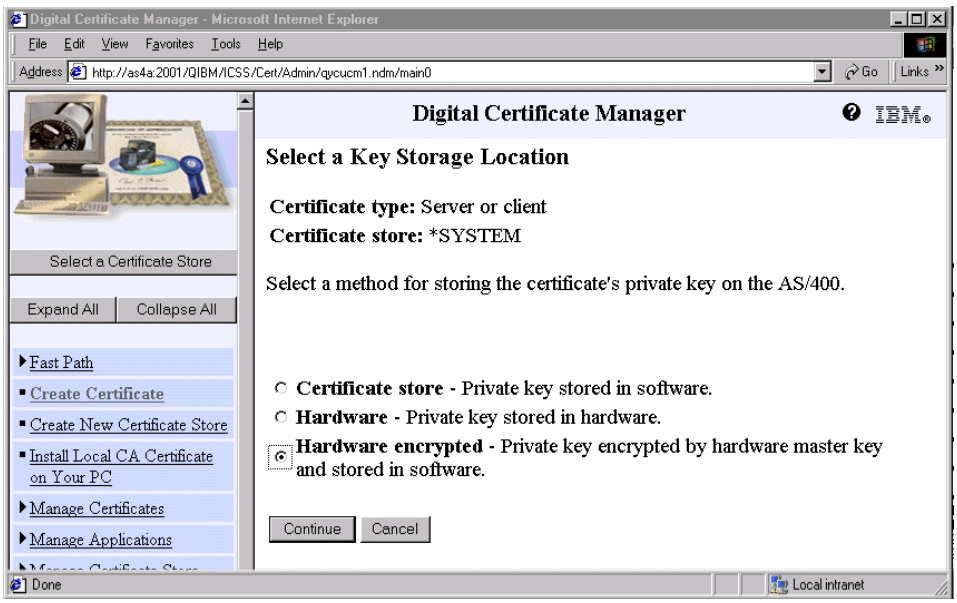


Figure 130. Select a Key Storage Location window

In the Select a Key Storage Location window, you decide where the private key of the new certificate will be stored. In order for the 4758 Cryptographic Coprocessor to perform SSL-related tasks, you need to specify **Hardware** or **Hardware encrypted**; otherwise the certificate cannot be used by the coprocessor. For a detailed explanation of the various options and their implications, refer to 4.4.1, “What you should consider first” on page 206.

In this scenario there are already two 4758 Cryptographic Coprocessors installed in the system and we want to share the load between these coprocessors. Therefore, we selected the Hardware encrypted option.

10. Select **Hardware encrypted** and click **Continue**.

Figure 131. Select a Cryptographic Device Description window

The Select a Cryptographic Device Description window lets you specify which 4758 Cryptographic Coprocessor will be used to generate the private key. If you specified in the previous step that the private key will be stored in software protected by hardware, the coprocessor after creating the private key encrypts it with the master key and stores it in the key store file. When hardware was selected, the private key is stored on the coprocessor adapter itself.

If you have multiple coprocessors installed in your system and they are grouped, which means several coprocessors share the same key store file, you can select any coprocessor within the group to create the private key. If you have independent groups, you have to make sure to pick a coprocessor from the correct group.

11. Select the device description, in this scenario **CRP01,** and click **Continue**.

Figure 132. Select Additional Cryptographic Device Descriptions window

This window shows all configured 4758 Cryptographic Coprocessors except the one that is used to generate the private key. If you want to share the load between multiple coprocessors, you can select the ones that can perform SSL handshake processing for this particular certificate. In case you want to start with one coprocessor and expand later, you can do this by updating the device assignment as described in 4.6, “Updating the 4758 Cryptographic Coprocessor device assignment” on page 218.

For more information about load sharing and its prerequisites, refer to 4.7, “Load sharing” on page 224.

12. Select **CRP02** or, if more coprocessors are available, all which should be used for load sharing.
13. Click **Continue**.

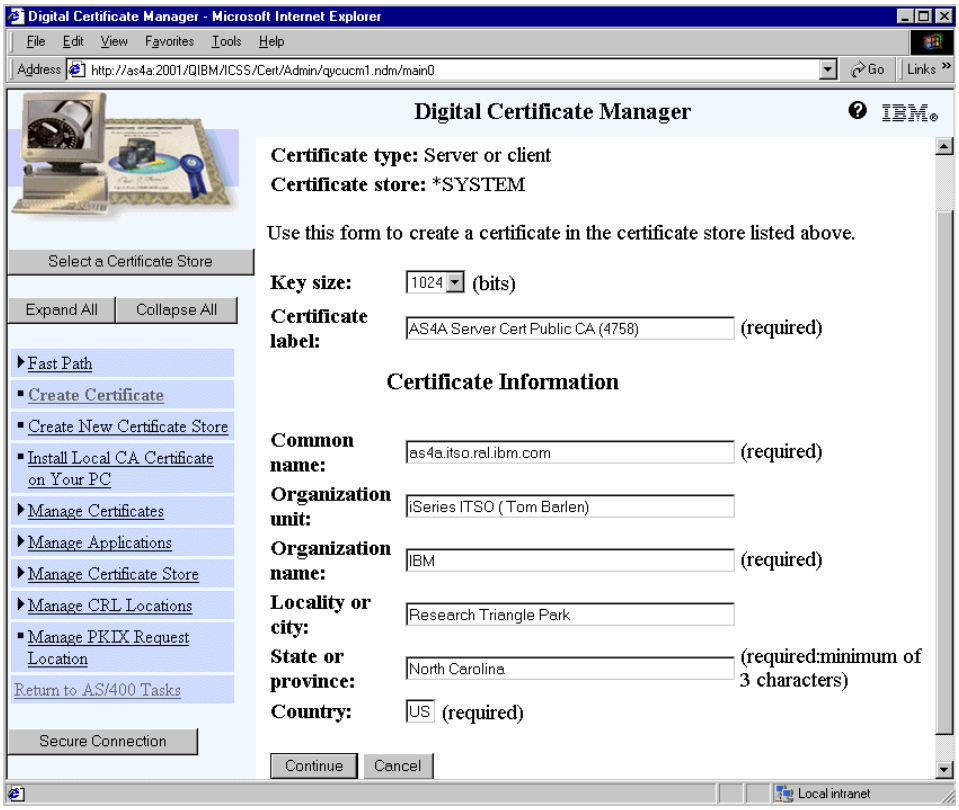


Figure 133. Create Certificate window

Fill out the certificate request data that makes up the subject's distinguished name (DN). To easily recognize a certificate that is being used by the 4758 Cryptographic Coprocessor you can add a comment, such as *(4758)* to the certificate label as shown in Figure 133. The available key sizes for the public/private key pair depend on the installed Cryptographic Access Provider product.

14. Click **Continue** to create the certificate request.

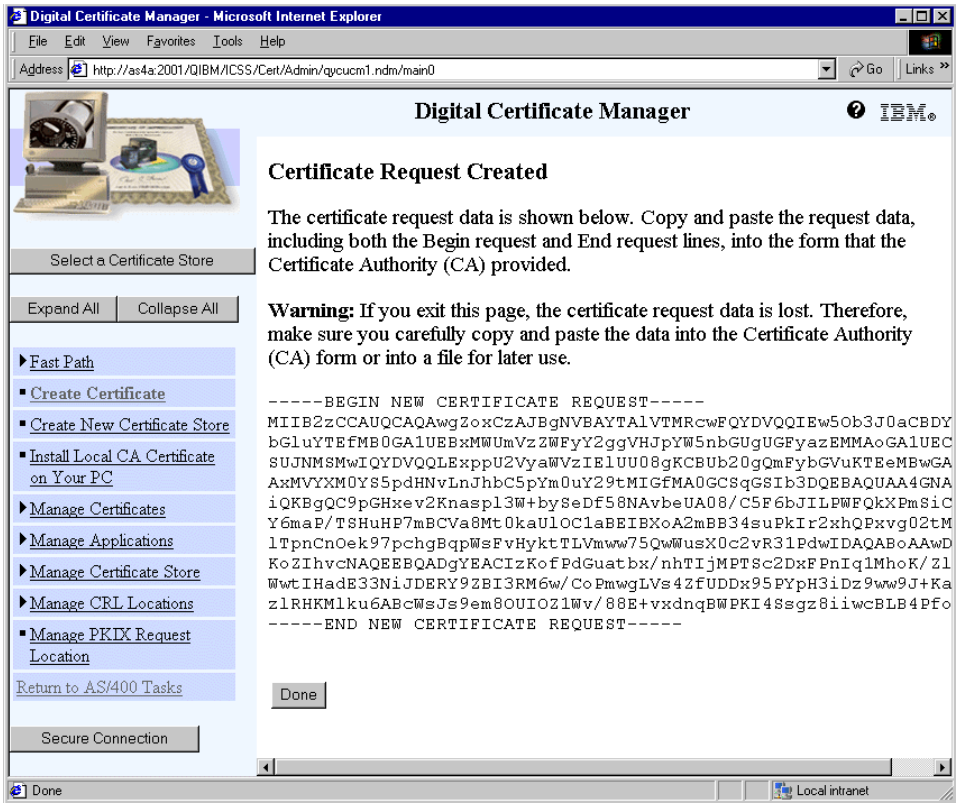


Figure 134. Certificate Request Created window

After the public/private key pair is created, DCM creates a Certificate Signing Request (CSR) which contains the subject's DN along with the public key. The CSR is then displayed in Base64 encoded form on the DCM window. You now need to copy and paste the information into a file or Web site of the CA (for example, VeriSign or Thawte). The data to be copied must include the `-----BEGIN NEW CERTIFICATE REQUEST -----` and the `----- END NEW CERTIFICATE REQUEST -----` tags.

15. Click **Done**.

    The CA will verify the certificate request and will sign it with its private key and then send the signed certificate back. This request may take some time. After you received the signed certificate, store it in an IFS directory on the iSeries server.

16. Start DCM and open the *SYSTEM certificate store.

17. On the navigation pane, click **Fast path**.

18. Select **Work with server and client certificates** and click **Continue**.
19. A list of certificates is displayed. Click **Import** on the bottom of the window to import the signed certificate.
20. When prompted specify the IFS directory and file name that contains the signed certificate and click **Continue**.
21. Assign the new certificate to the applications that will use the certificate for SSL authentication.

This completes the process of requesting, importing, and configuring a certificate in DCM that will be used by the 4758 Cryptographic Coprocessor.

4.5 Coprocessor device assignment for a given certificate

This section describes the steps to determine whether a certificate's private key is stored in a certificate store, on the 4758 Cryptographic Coprocessor, or in a key store file protected by the master key of a cryptographic coprocessor. It also shows you how to find out which certificate is assigned to a 4758 Cryptographic Coprocessor.

1. Start the AS/400 Tasks page using a Web browser and enter the following URL:

   `http://servername:2001`

   Where *servername* represents your iSeries host name. The options available from the AS/400 Tasks page vary depending on the installed program products.
2. On the AS/400 Tasks page, click **Digital Certificate Manager** to start DCM. Refer to Chapter 2, "Digital Certificate Manager" on page 11, for more information about using DCM.

   The DCM window appears. It is separated into two parts: the major pane on the right and the navigation pane on the left which depending on the selected store and user authority, shows the available options for the user signed on to DCM.
3. From the navigation pane, click **Select a certificate store**.
4. Select the ***SYSTEM** certificate store and click **Continue**.
5. Enter the certificate store password and click **Continue**.
6. On the navigation pane, click **Fast Path**.
7. Select **Work with server and client certificates** and click **Continue**.

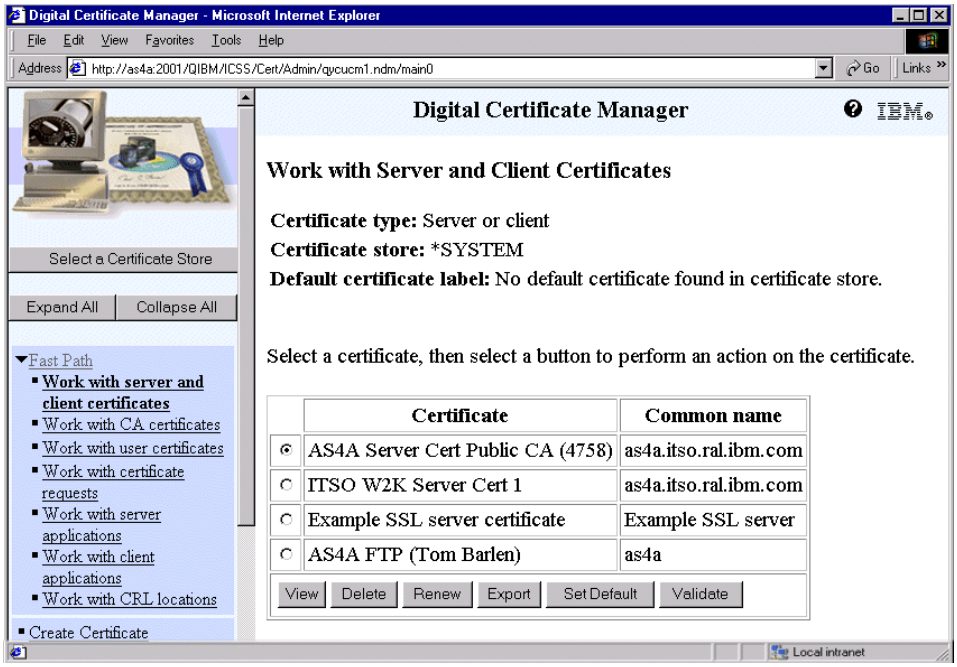


Figure 135. Work with Server and Client Certificates window

The Work with Server and Client Certificates window lists all server and client certificates in the *SYSTEM certificate store. Let us assume you do not know if a certain certificate is used by a 4758 Cryptographic Coprocessor, but you want to know if it is. You just need to select a certificate and view its properties.

8. Select a certificate (in this example it is the first one in the list) and click **View**.

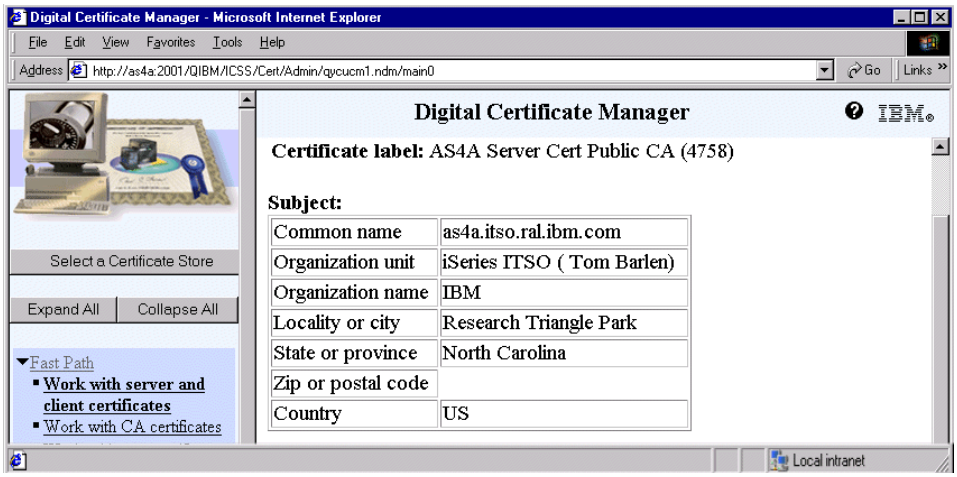


Figure 136. Viewing a server certificate window (Part 1)

There is a lot of information associated with a certificate. When viewing certificate data, the listed information starts with the certificate's subject distinguished name (DN) and the continues with an additional information section where you find information, such as whether a private key is available for this certificate or the serial number.

9. Scroll down the window until you reach the section headed *Private key information*.

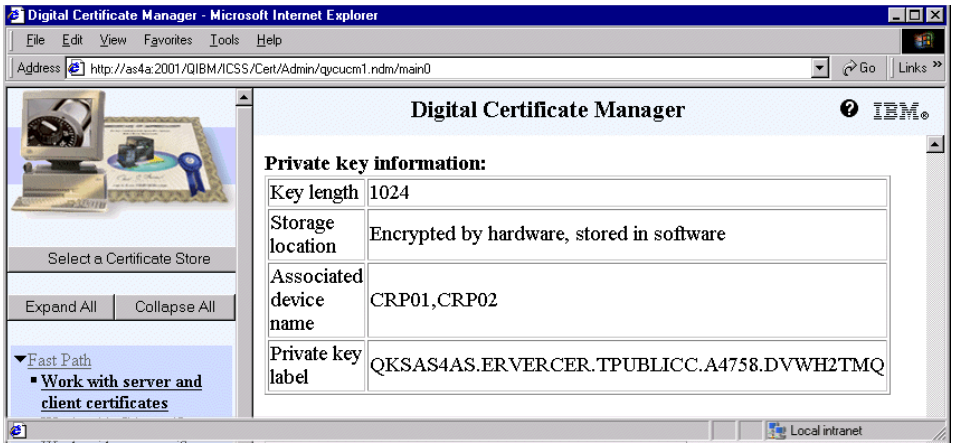


Figure 137. Viewing a server certificate window (Part 2)

The section Private key information contains the essential data you need to tell whether a certificate is being used by a 4758 Cryptographic Coprocessor.

The two most important fields are:

Storage location: This field provides information on where the certificate's private key is stored. If the value is *Encrypted by hardware, stored in software* the private key is stored in a key store file encrypted by the cryptographic coprocesor's master key. The certificate can then be used for hardware cryptography SSL support. If the value states *Stored in hardware* then the private key is stored on the coprocessor adapter itself and can also be used only by the 4758 Cryptographic Coprocessor that contains the private key. The third value is *Stored in software* and means that the certificate's private key is stored in the *SYSTEM certificate store. In this case the 4758 Cryptographic Coprocessor cannot be used for this certificate.

Associated device name: This field lists all cryptographic coprocessor devices that are assigned to the certificate. In the example shown in Figure 137, the coprocessor's CRP01 and CRP02 are assigned to use this particular certificate.

The Private key label field contains the label used by the 4758 Cryptographic Coprocessors to identify the private key on the adapter or the key store file.

Both the Associates device name and the Private key label fields are not shown when the private key is stored in the *SYSTEM certificate store.

The View Applications button at the bottom of the certificate information window allows you to display all secure applications that have this certificate assigned to.

10. Click **OK** to return to the Work with Server and Client Certificates window.

4.6 Updating the 4758 Cryptographic Coprocessor device assignment

Certain circumstances, such as adding an additional 4758 Cryptographic Coprocessor for load-sharing purposes, require you to update the cryptographic coprocessor device assignment. This section guides you through the steps of changing or updating the cryptographic coprocessor device assignment for a given certificate. However, updating a device

assignment only makes sense, is and only supported, if you have more than one 4758 Cryptographic Coprocessor installed and the certificate's private key is stored in a key store file protected by the coprocessor's master key. Perform the following steps to update a device assignment:

1. Start the AS/400 Tasks page using a Web browser and enter the following URL:

   `http://servername:2001`

   Where *servername* represents your iSeries host name.

2. On the AS/400 Tasks page click **Digital Certificate Manager** to start DCM. Refer to Chapter 2, "Digital Certificate Manager" on page 11 for more information about using DCM. The DCM window appears. It is separated into two parts: the major pane on the right and the navigation pane on the left, which depending on the selected store and user authority, shows the available options for the user signed on to DCM.
3. From the navigation pane, click **Select a certificate store**.
4. Select the ***SYSTEM** certificate store and click **Continue**.
5. Enter the certificate store password and click **Continue**.
6. On the navigation pane, expand **Manage Certificates** and click **Update device assignment**. You can also start the update device assignment task via the Fast Path navigation option, but this method requires additional steps to reach the same point.

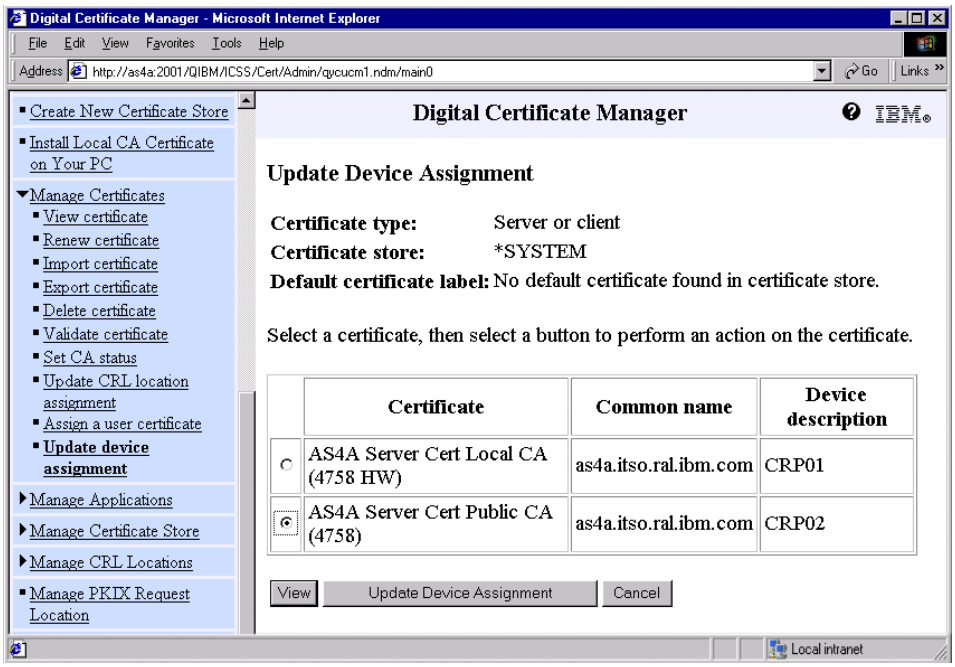


Figure 138. Update Device Assignment window

The Update Device Assignment window lists all certificates that have private keys stored in the 4758 Cryptographic Coprocessor or stored in a key store file protected by the coprocessor's master key. The information displayed contains the certificate label, the certificate's common name, and the cryptographic coprocessor devices that are currently assigned to the certificates listed.

7. Select the certificate you want to update the device assignment for, in this example the certificate with the label *AS4A Server Cert Public CA (4758)*.
8. Click **Update Device Assignment**.

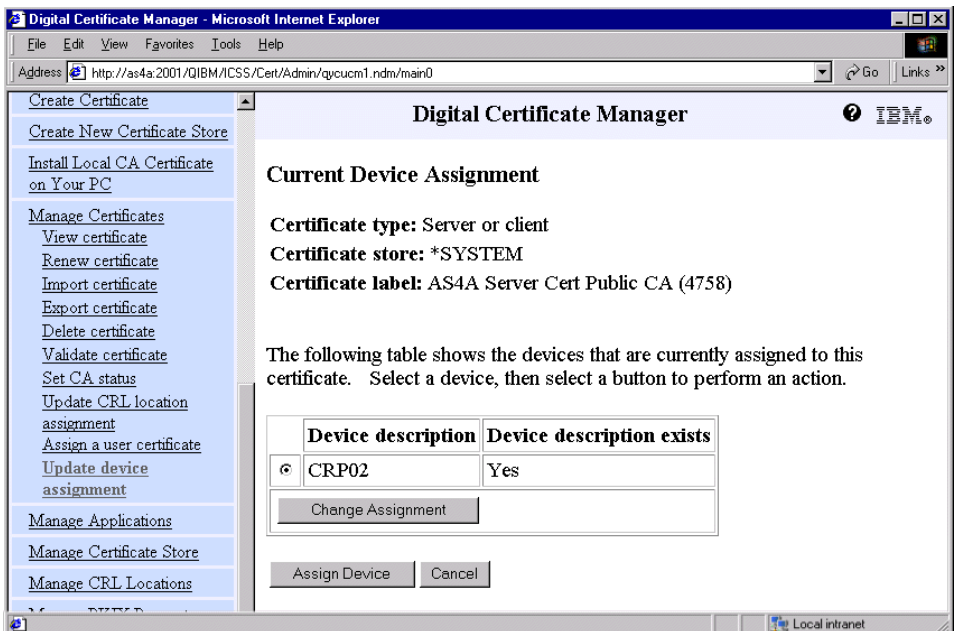


Figure 139. Current Device Assignment window

From the Update Device Assignment window you have the following options:

Change Assignment This option allows you to assign another coprocessor device to the certificate. The current assignment will be removed. Use the Assign Device option to assign an additional device to the certificate.

Remove Assignment If you have more than one cryptographic coprocessor device assigned to a certificate, this option can be used to remove a device assignment from the certificate. However, there must be at least one device assigned to a certificate, which means you cannot remove all devices from a certificate. This is also the reason why the Remove Assignment option is not shown in Figure 139.

Assign Device Use this option to assign additional cryptographic coprocessor devices to a certificate. For example, if you add another 4758 Cryptographic Coprocessor to the system and want to share the

load between the two coprocessors, you can use this option to assign the new coprocessor to the certificate.

As mentioned previously, you can update the device assignment for all certificates that have private keys stored in or protected by a 4758 Cryptographic Coprocessor. But the only option you have for certificates that have their private key stored in the hardware adapter is to navigate through the windows. The reason is that the private key cannot be exported from the coprocessor and therefore no update of a device assignment is possible.

9. Currently, the certificate has only the CRP02 coprocessor assigned. In this scenario we have experienced performance problems and decided to use an additional 4758 Cryptographic Coprocessor to share the load.

   Click **Assign Device**.

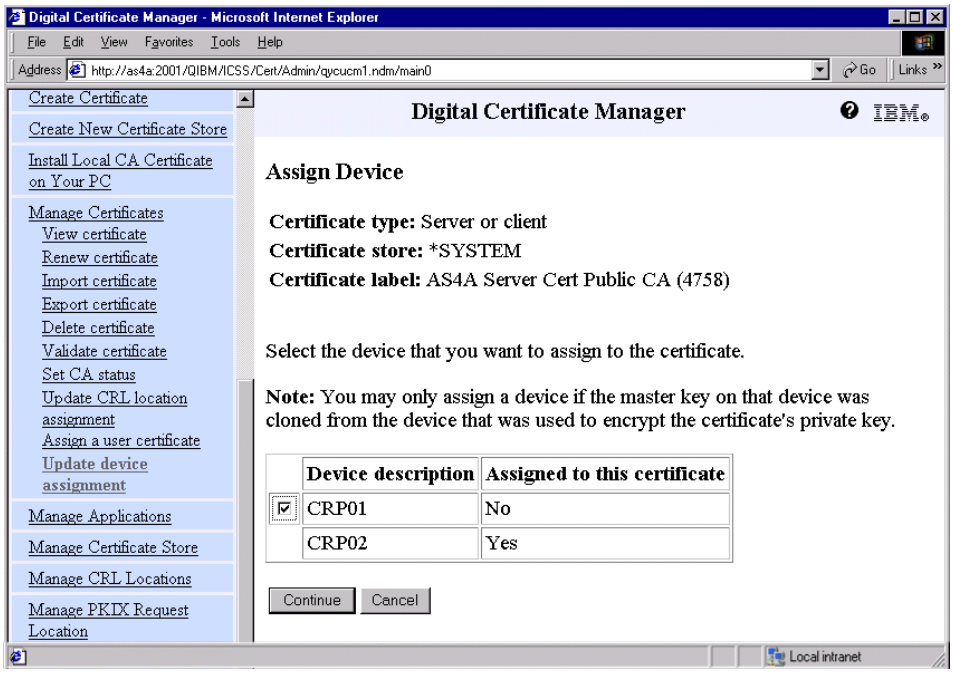


Figure 140. Assign Device window

Select the device you want to add to the certificate from the list of configured 4758 Cryptographic Coprocessors.

In this example, select the CRP01 device.

Important

When you assign an additional device to a certificate, for example, for load sharing, you will always get a successful completion message, even if the master keys of the assigned cryptographic coprocessor devices are different. DCM does not and cannot check if the master keys of both 4758 Cryptographic Coprocessors are the same. But this is a requirement for the coprocessors to access and use the certificate's private key. For more information about load sharing with the 4758 Cryptographic Coprocessor, refer to 4.7, "Load sharing" on page 224.

10. Click **Continue** to update the device assignment.

11. From the confirmation window click **OK** to return to the Current Device Assignment window.

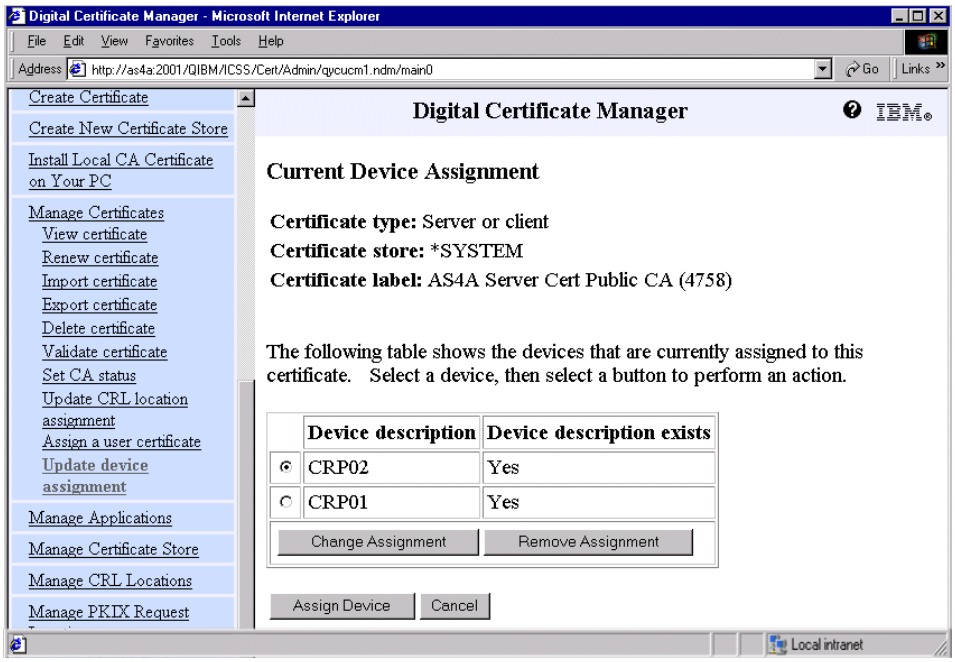


Figure 141. Current Device Assignment window after the update

Both devices are now assigned to the certificate. Note that the Remove Assignment option is now available.

12. Click **Cancel** to complete the task.

This concludes the update device assignment task.

4.7 Load sharing

Due to an extensive load, it may become necessary to use more than one 4758 Cryptographic Coprocessor for performing parts of the SSL handshake. In V5R1 the maximum number of supported 4758 Cryptographic Coprocessors were increased from three to eight coprocessors in a single iSeries server, depending on the available PCI slots for a particular iSeries model.

The implementation in OS/400 allows a user without additional programming effort to utilize load sharing between multiple cryptographic coprocessors for SSL applications managed by DCM. The load is shared across all cryptographic coprocessors that are assigned to the certificate used for SSL authentication. The system uses the round-robin method for assigning the work to an individual coprocessor. Let's take a closer look how the system manages load sharing. Only one coprocessor can be allocated to a single job (SSL-enabled application) at a given time, but the job can allocate and deallocate the devices so that several coprocessor devices can be used. For SSL-enabled applications that are managed in DCM, the allocation and deallocation is managed by the system, if the certificate indicates that more than one coprocessor is to be used for SSL session establishment.

A very important aspect of implementing load sharing is that all cryptographic coprocessors that will be assigned to a single certificate must have the same master key and they have to use the same key store file for storing the private keys.

4.7.1 Requirements for load sharing

This section provides a summary of tasks to be performed to set up load sharing for 4758 Cryptographic Coprocessors when used to off-load SSL session establishment work from the main processor.

4.7.1.1 Implementing load sharing in stages

The following list describes the steps to be performed when you start with one cryptographic coprocessor to be used for SSL and want to be able to implement load sharing at a later time.

1. Install the hardware and software prerequisites mentioned in 4.1.1, "Hardware requirements" on page 191, and 4.1.2, "Software requirements" on page 192.

2. Configure the first 4758 Cryptographic Coprocessor as described in 4.3, “Configuring the 4758 Cryptographic Coprocessor” on page 195.
3. Request a new certificate and specify the option to store the private key in software protected by hardware. Refer to 4.4, “Configuring DCM to utilize hardware cryptography for SSL” on page 206.

The first three tasks enable the hardware cryptography support for SSL session establishment. The remaining tasks have to be performed when installing an additional cryptographic coprocessor to share the load with the first one.

4. Install and configure the new 4758 Cryptographic Coprocessor as described in 4.3, “Configuring the 4758 Cryptographic Coprocessor” on page 195. Remember to use the same key store file (PKA key file) as the first cryptographic coprocessor. If you used the basic configuration wizard to configure the first coprocessor and you wrote down the master key, you can enter this master key during configuration of the new coprocessor. This saves you from having to perform the next step, the cloning process. However, writing down a key or password might compromise your security policies.
5. If you do not know the master key of the first coprocessor, you have to clone the master key to the new coprocessor as described in 4.8, “Cloning the master key” on page 226.
6. Update the device assignment for the certificates that are associated with SSL-enabled applications you want to have load sharing for. Refer to 4.6, “Updating the 4758 Cryptographic Coprocessor device assignment” on page 218, for detailed instructions.

4.7.1.2 Implementing load sharing in one step

The following summary lists the tasks to be performed when you have not used any 4758 Cryptographic Coprocessor for SSL and plan to implement load sharing with at least two coprocessors the first time.

1. Install the hardware and software prerequisites mentioned in 4.1.1, “Hardware requirements” on page 191, and 4.1.2, “Software requirements” on page 192.
2. Configure all the 4758 Cryptographic Coprocessors as described in 4.3, “Configuring the 4758 Cryptographic Coprocessor” on page 195. Remember to use the same key store file (PKA key file) as the first cryptographic coprocessor. If you used the basic configuration wizard to configure the first coprocessor and you wrote down the master key, you can enter this master key during configuration of the new coprocessor. This saves you from having to perform the next step, the cloning process.

However, writing down a key or password might compromise your security policies.

3. If you do not know the master key of the first coprocessor, you have to clone the master key to the second coprocessor as described in 4.8, “Cloning the master key” on page 226.
4. Request a new certificate and specify the option to store the private key in software protected by hardware. Make sure you also select all cryptographic coprocessors that are to be used for load sharing with the new certificate. Selecting all coprocessors during certificate creation time saves you from having to perform the device assignment update task. Refer to 4.4, “Configuring DCM to utilize hardware cryptography for SSL” on page 206.

4.8 Cloning the master key

The master key is configured on a 4758 Cryptographic Coprocessor during the initial setup. It is also called a key-encrypting key that has a length of 168 bits. The master key is used to encrypt other keys, such as private keys (PKA keys), that are stored outside the 4758 Cryptographic Coprocessor adapter. An example of storing a private key outside of the coprocessor adapter is when you use DCM to create a certificate specifying that the private key is stored in software protected by hardware. In this case the private key is stored in a key store file encrypted by the coprocessor’s master key. When the coprocessor needs to perform tasks that require the externally stored key, it retrieves the encrypted key from the key store file and decrypts it with the master key inside the coprocessor hardware.

If you want to use multiple 4758 Cryptographic Coprocessors to share the load, you need to have the same master key on all coprocessors that are assigned to a particular certificate. That means if you assign a coprocessor device to a certificate with a different master key from the coprocessor that was used to generate the private key and a SSL connection request is to be processed, the newly assigned coprocessor will not be able to decrypt the private key. Thus, the SSL handshake will fail.

The most secure way to transmit the master key from one coprocessor to another coprocessor is use the cloning process. If you decided to write down the master key when you configured the first coprocessor, you can also enter this master key during the configuration of another coprocessor. In this case you do not use the cloning process. However, be aware that writing down a password or master key can compromise your security policies.

In highly secure environments, you might have chosen during the setup of the 4758 Cryptographic Coprocessor that the master key is to be auto-generated. Using this method, the master key is not displayed on any configuration window, nor can it be retrieved from the coprocessor. In other words, the master key cannot be written down and stored in a secure place. But what are you going to do when the cryptographic coprocessor adapter fails? All keys that are stored in a key store file protected by the master key could not be recovered anymore and would be lost. For these cases, cloning of the master key could be used to back up the master key onto another coprocessor, allowing you to recover the externally stored keys.

4.8.1 How does cloning work?

The cloning process securely transmits the master key without exposing the value of the master key from one 4758 Cryptographic Coprocessor to another. The process splits the master key into *n* shares, where *n* is a number from 1 to 15. *m* shares are required to rebuild the master key in another coprocessor, where *m* is a number from 1 to 15 and less than or equal to *n*.

Let us use an example to explain the relation between *m* and *n* shares. Assume you want to split the master key into eight shares (*n*) and you specify that you need six shares (*m*) to rebuild the master key. The system then distributes the split master key in a way that it stores the information into eight different files (shares), but any six of the files contain enough information to rebuild the key. The process of using *n* and *m* shares to securely transmit the information is based on Shamir's secret sharing algorithm. For more information about this algorithm, refer to the RSA Web site at `http://www.rsasecurity.com/`.

When operating the 4758 Cryptographic Coprocessor in the most secure manner, no one person (profile) is allowed to perform all the hardware commands necessary to clone the entire master key. In this case you can set up the coprocessor so that, for example, you configure four profiles on the coprocessor, but only three are required to rebuild the master key. You would then authorize each profile to perform installation steps only for two shares. The following table shows the authorization scheme that is needed in this example to receive the shares.

Table 5. Hardware command authorization for creating and receiving shares

4758 Cryptographic Coprocessor hardware command authorization			
Profile A	**Profile B**	**Profile C**	**Profile D**
Clone Info Install 1	Clone Info Install 3	Clone Info Install 5	Clone Info Install 7

4758 Cryptographic Coprocessor hardware command authorization			
Profile A	**Profile B**	**Profile C**	**Profile D**
Clone Info Install 2	Clone Info Install 4	Clone Info Install 6	Clone Info Install 8

According to the information provided in this example, eight shares are created, but only six are required to rebuild the master key. The following combination of profiles can be used to rebuild the master key:

- Profiles A, B, and C
- Profiles B, C, and D
- Profiles A, C, and D
- Profiles A, B, and D

The advantage of setting up an environment as described earlier is that even if one person with its profile is not present at the time of cloning, the other three are able to rebuild the key. For a listing of the available hardware commands refer to Appendix A, “4758 cryptographic coprocessor hardware commands” on page 403.

The coprocessor containing the master key to be cloned is referred to as the Sender. The Sender must generate an RSA key pair where the private key is stored on the Sender coprocessor as a retained key. The public key is stored in a file. The private key must also be marked as suitable for use with cloning when it is generated. The key is known as the Sender key. The coprocessor that will receive the master key is referred to as the Receiver. The Receiver must also generate a retained RSA key pair. The private key, which is referred to as the Receiver key, must also be marked as suitable for use with cloning.

Both the Sender and Receiver public keys must be digitally signed or certified by another retained private key in a coprocessor, referred to as the Certifier. This retained private key is the Certifier key. The associated public key must be registered in both the Sender and the Receiver before shares can be generated and received. A 4758 Cryptographic Coprocessor can take on the role of Certifier only, or can be both Certifier and Sender, or it can be both Certifier and Receiver.

As each share is generated, it is encrypted by a newly generated triple-DES key and signed by the coprocessor using the Sender private key. The triple-DES key is then encrypted by the Receiver public key.

As each share is received, the signature on the share is verified using the Sender public key, the triple-DES key is decrypted using the Receiver private

key, and the share decrypted using the triple-DES key. When *m* number of shares have been received, the cloned master key will be complete within the new master key register of the Receiver.

After the master key is loaded into the coprocessor, you need to set the new master key to activate it.

4.8.2 Performing the master key cloning process

The easiest way of cloning the master key is by using the 4758 Cryptographic Coprocessor configuration wizard. But even with using the wizard, the cloning process still involves many steps to be performed. It is very important that you carefully follow the documented steps. We recommend to even write a log of what steps you performed at which time. It is very easy to accidently select the wrong coprocessor to work with and the result is that you perform certain tasks on the wrong coprocessor. In the worst case you would even clone the wrong master key, for example, from the new coprocessor to the production coprocessor and thus would lose access to keys.

4.8.2.1 The cloning scenario

In this scenario, a company is already using a 4758 Cryptographic Coprocessor to improve SSL session establishment performance for their e-Business applications. Due to an extensive growth of their business, they experience performance problems. A performance analysis has revealed that the cryptographic coprocessor is the bottleneck.

The current installation as shown in Figure 142 is using one cryptographic coprocessor.

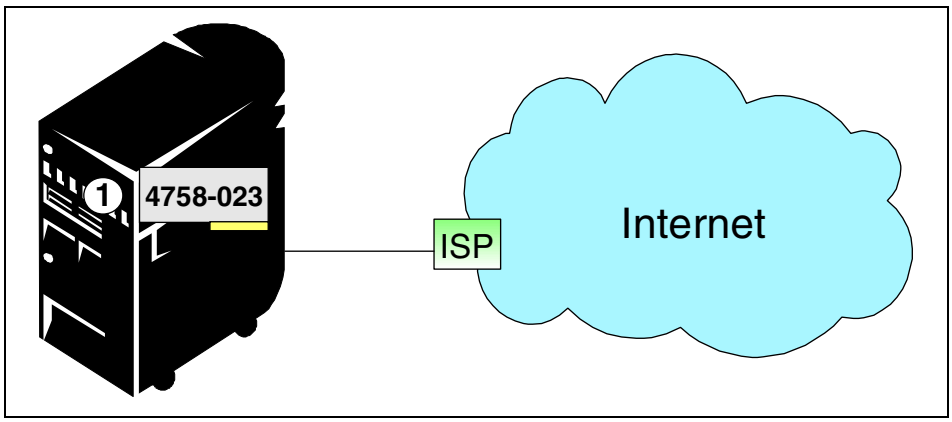


Figure 142. Current installation with one 4758 Cryptographic Coprocessor

It turned out that a second 4758 Cryptographic Coprocessor would be required to improve performance. The new coprocessor would need to be set up so that the iSeries server can share the load between the existing and the new coprocessor. Figure 143 illustrates the system configuration after adding the second cryptographic coprocessor.

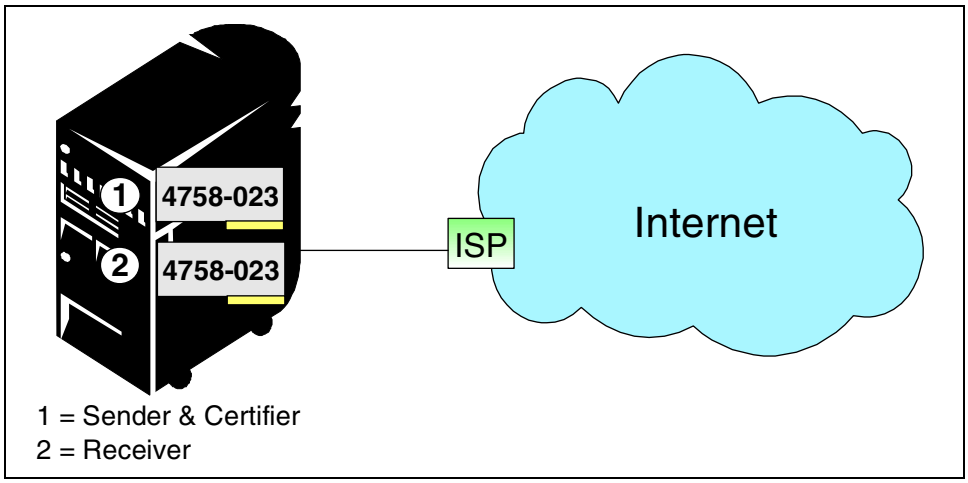


Figure 143. New installation with two 4758 Cryptographic Coprocessors

As shown in Figure 143, the 4758 Cryptographic Coprocessor (1) represents the existing coprocessor. It contains the master key to be cloned and has in this scenario the role of the Sender.

The new 4758 Cryptographic Coprocessor (2) will receive the master key from coprocessor 1 and therefore acts as the Receiver.

The third role involved in cloning the master key is the Certifier role. If you have more than two 4758 Cryptographic Coprocessors installed in the system you can use a third coprocessor for better security as the certifier. In case you have only two coprocessors, as it is in this scenario, one of the two has to be the certifier. We decided to use the Sender coprocessor to also perform the Certifier tasks. For more information about the roles involved and how the cloning process works, refer to 4.8.1, “How does cloning work?” on page 227.

Important

The coprocessor you want to clone the master key to (Receiver) has to be configured first before starting the cloning process. The new coprocessor's device description has to be configured to use the same PKA key store file than the existing coprocessor. Refer to 4.3, "Configuring the 4758 Cryptographic Coprocessor" on page 195, for information on how to perform the coprocessor's basic configuration.

Since it is very easy to lose track of the various input parameters during the cloning process, we have decided to provide you with a table that contains the values we used in this scenario. If you perform master key cloning, you can use the same values as listed in Table 6. The only values you might want to change are the number of shares (*n* and *m)* and the directory you will use to store the public keys and the files containing the shares.

Table 6. Input parameter and values for master key cloning

Step	Copro cessor	Role	Function	Parameter	Value	Reference to step in process
1	1	Sender	Set N & M	- Max. number shares N - Min. number shares M	8 6	9. on page 237
2	1	Sender	Create Sender RSA keys	- Private key name - Stream file to store public key	sndprv /clone/sndpub	13. on page 239
3	2	Receiver	Set N & M	- Max. number shares N - Min. number shares M	8 6	23. on page 244
4	2	Receiver	Create Receiver RSA keys	- Private key name - Stream file to store public key	rcvprv /clone/rcvpub	26. on page 245
5	1	Certifier	Create share certification RSA keys	- Private key name - Stream file to store public key	shrprv /clone/shrpub	36. on page 249
6	1	Certifier	Sign Sender's public key	- Stream file containing Sender's public key - Certifier private key	/clone/sndpub shrprv	39. on page 250
7	1	Certifier	Sign Receiver's public key	- Stream file containing Receiver's public key - Certifier private key	/clone/rcvpub shrprv	42. on page 251

Step	Copro cessor	Role	Function	Parameter	Value	Reference to step in process
8	1	Sender	Register hash of Certifier's public key	- Share certification public key name - Stream file containing the certifying key	shrpub /clone/shrpub	47. on page 253
9	1	Sender	Register key	- Share certification public key name - Stream file containing the certifying key	shrpub /clone/shrpub	50. on page 254
10	1	Sender	Get share (step repeated for number of N shares)	- Share to process - Sender private key name - Certification public key - Stream file containing Receiver's public key - Stream file to store share	01 sndprv shrpub /clone/rcvpub /clone/share1	53. on page 256
11	2	Receiver	Register hash of Certifier's public key	- Share certification public key name - Stream file containing the certifying key	shrpub /clone/shrpub	67. on page 260
12	2	Receiver	Register key	- Share certification public key name - Stream file containing the certifying key	shrpub /clone/shrpub	70. on page 261
13	2	Receiver	Receive shares (step repeated for number of M shares)	- Share to receive - Receiver private key name - Certification public key - Stream file containing Sender's public key - Stream file containing share	01 rcvprv shrpub /clone/sndpub /clone/share1	73. on page 263
14	2	Receiver	Set master key	- Master key to be set	Both	79. on page 266

4.8.2.2 Cloning step by step

All 4758 Cryptographic Coprocessor configuration tasks have to be performed using a secure session. If you have not set up the ADMIN server instance for SSL, do it know. For information about enabling SSL for the ADMIN server instance refer to Appendix C, "Enabling SSL for the ADMIN

server instance" on page 415. Perform the following steps to clone the master key from the Sender coprocessor (1) to the Receiver coprocessor (2):

1. Start the AS/400 Tasks page using a Web browser and enter the following URL:

   `http://servername:2001`

   Where `servername` represents your iSeries host name. The options available from the AS/400 Tasks page vary depending on the installed program products.

2. After signing on to the AS/400 Tasks page, click **4758 Cryptographic Coprocessor**.

3. Click **Start secure session** to launch the 4758 Cryptographic Coprocessor configuration window. Note that you do get this option if you started the AS/400 Tasks page over a secure connection (port 2010).

4. On the navigation bar, click **Manage configuration** to expand its options and click **Master keys**.

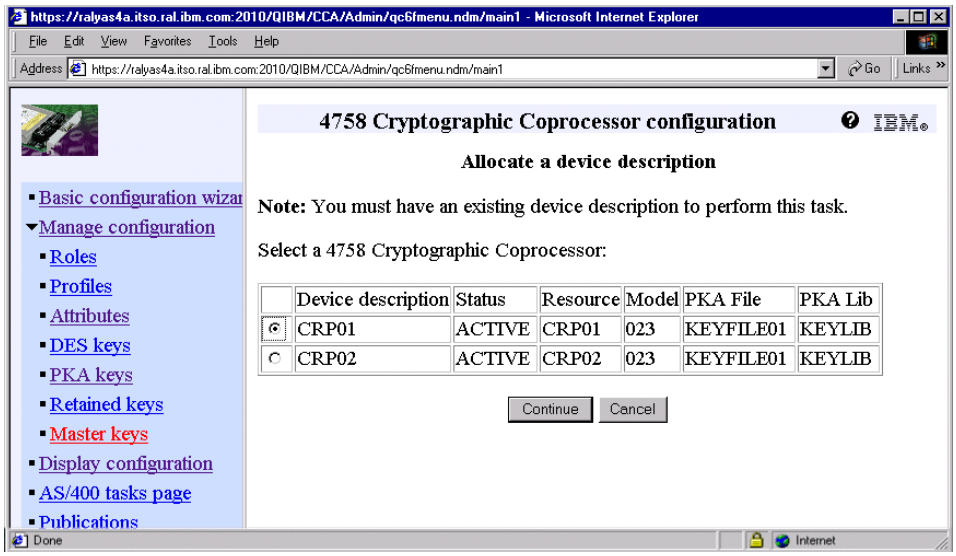


Figure 144. Allocate a device description window

5. Select **CRP01** and click **Continue**. This is the 4758 Cryptographic Coprocessor with the master key to be cloned (Sender). If the coprocessor is varied off, the system automatically varies it on.

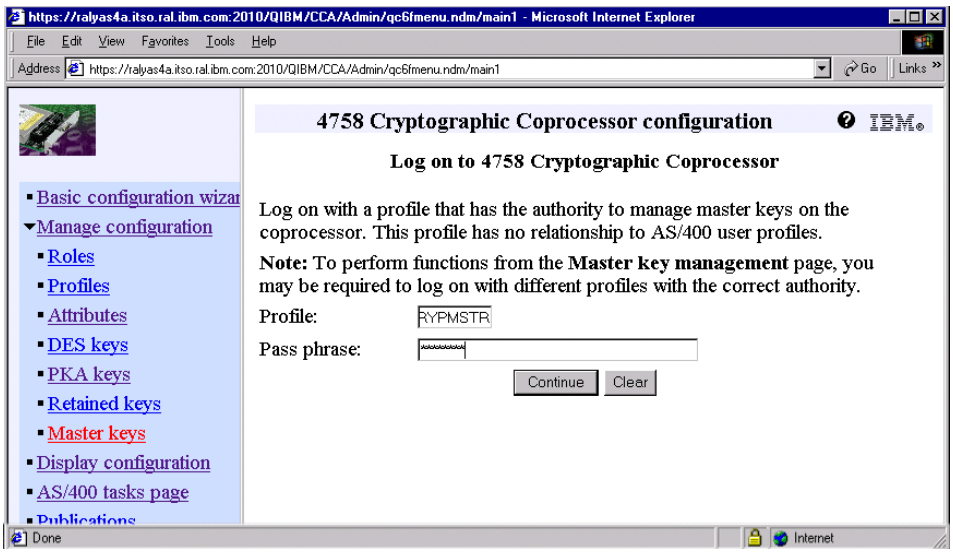


Figure 145. Log on to 4758 Cryptographic Coprocessor window

Enter CRYPMSTR as the profile and the profile's pass phrase. The profile and pass phrase information is case sensitive. Even if you enter a wrong or non-existing profile you will not receive an error message at this time. All authorizations will be checked when actually performing a coprocessor command. So you will receive an error message at a later time.

In this case the CRYPMSTR profile is used, since it has all the authorities needed to perform master key cloning.

6. Click **Continue**.

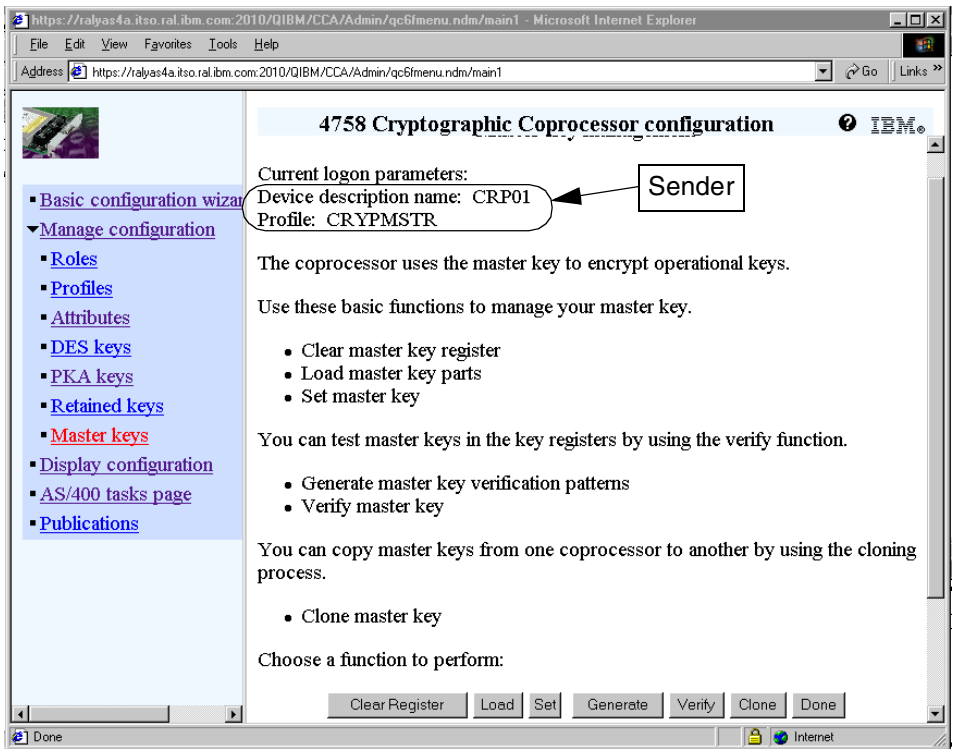


Figure 146. Master key management window

This window is the main window for all master key-related tasks. Clear Register allows you to clear the register of the new and the old master key; the current key cannot be cleared. The Load option is used to load a new master key, manually or automatically generated. With the Set option you can activate the new master key, which copies the new master key from the new master key register to the current master key register. The current master key is copied into the old master key register. The Generate and Verify options are used to determine if someone has altered the key.

7. Click **Clone** to display the master key cloning advisor.

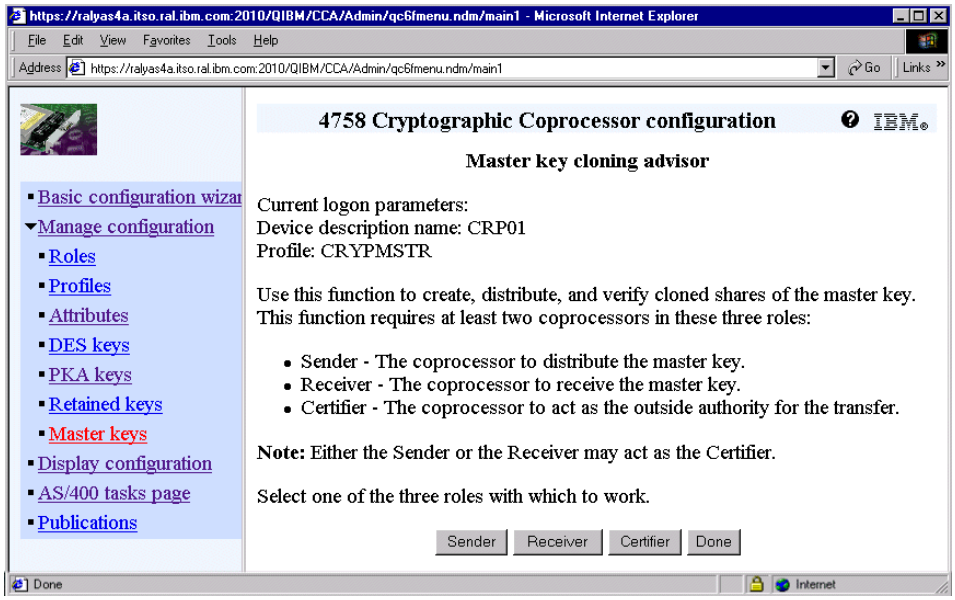


Figure 147. Master key cloning advisor window

The Master key cloning advisor window is the entry point for performing all cloning-related tasks.

8. Click **Sender**.

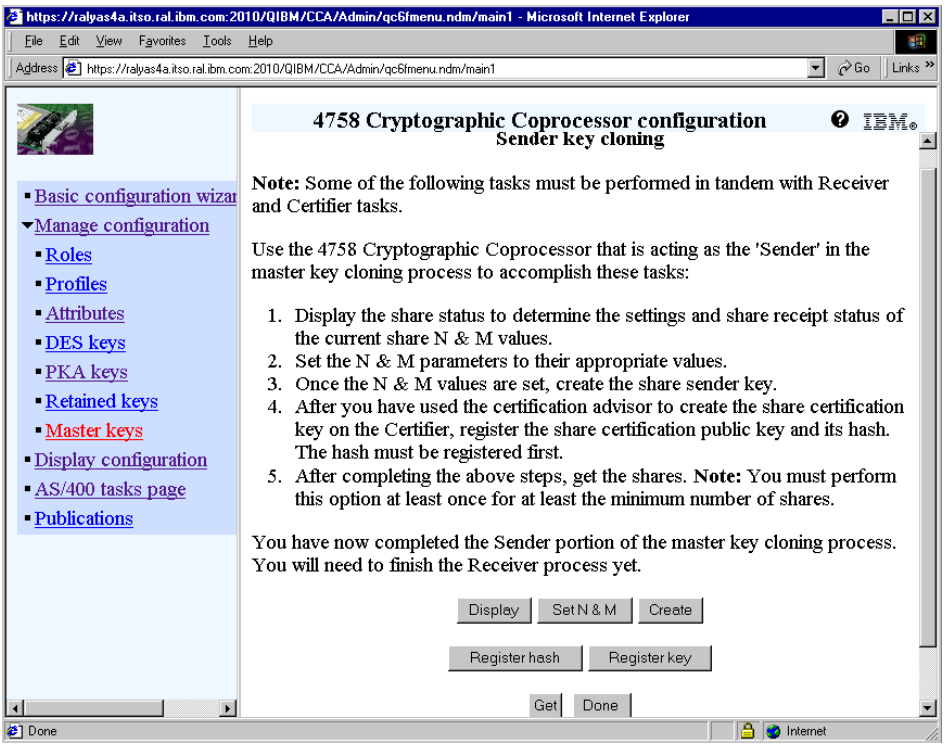


Figure 148. Sender key cloning window

The Sender key cloning window gives an overview of the steps that need to be performed to create the shares.

Do *not* click Display at this time, because there are no shares at the moment to display. If Display is clicked at this stage an error message is displayed stating that there is a problem with the key shares.

9. Click **Set N & M** to set the number of shares.

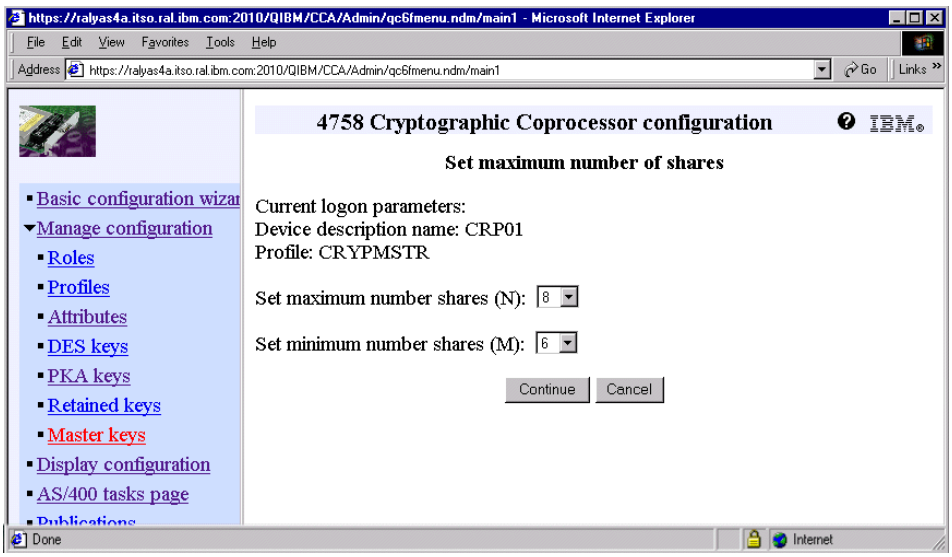


Figure 149. Set maximum number of shares window

The master key will be split into M shares but just N shared are used to rebuild the key. This allows you to share the cloning process to different people and let you design a highly secure environment. For more information on how shares are used within the cloning process, refer to 4.8.1, “How does cloning work?” on page 227.

10. Set the maximum number of shares (N) and the minimum number of shares (M). In this scenario we have decided to use 8 number of shares for N and 6 number of shares for M. Depending on security needs you may choose to work with more or less shares.

    Note that the number of shares for N and M must be the same on the Sender and the Receiver site.

    The more shares you want to use for cloning the master key the more steps you have to perform until you have the master key loaded on the second coprocessor. For example, if you choose to work with 14 N shares and 12 M shares, you have to perform 14 times the step to create a share on the Sender site and 12 times to receive the shares on the Receiver site. Again, the number of shares is directly related to the amount of work you have to perform, but also to how secure the cloning process is. The more shares, the more secure.

11. Click **Continue**.

    A confirmation message is displayed confirming that the share values have been successfully set.

12. Click **OK** on the confirmation message window to return to the Sender key cloning window.

13. From the Sender cloning window click **Create** to create the private and public Sender keys.

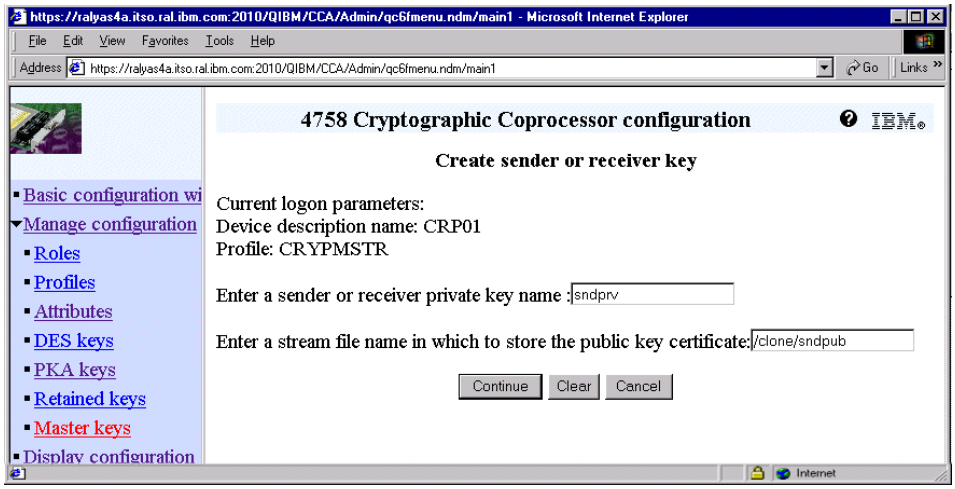


Figure 150. Create sender or receiver key window

Set the name for the Sender's private key and indicate where the public key will be stored. Make sure the directory where the public key has to be stored exists prior to this action.

- Sender private key name: `sndprv`
- Stream file to store the Sender's public key: `/clone/sndpub`

To keep the cloning process as simple as possible, we recommend that you store all cloning related files in the same directory. In this scenario, the clone directory is used.

14. Click **Continue**.

A confirmation message shows up indicating that the key was created successfully.

15. Click **OK** to close the confirmation message window.

You return to the Sender key cloning window.

The cloning tasks mentioned in the Sender key cloning window suggest that you continue with creating the Certifier first. In the cloning process we describe in this chapter, the Receiver environment is set up next and then the Certifier, which saves us one step.

16. Click **Done** on the Cloning key Sender window to return to the Master key cloning advisor window.

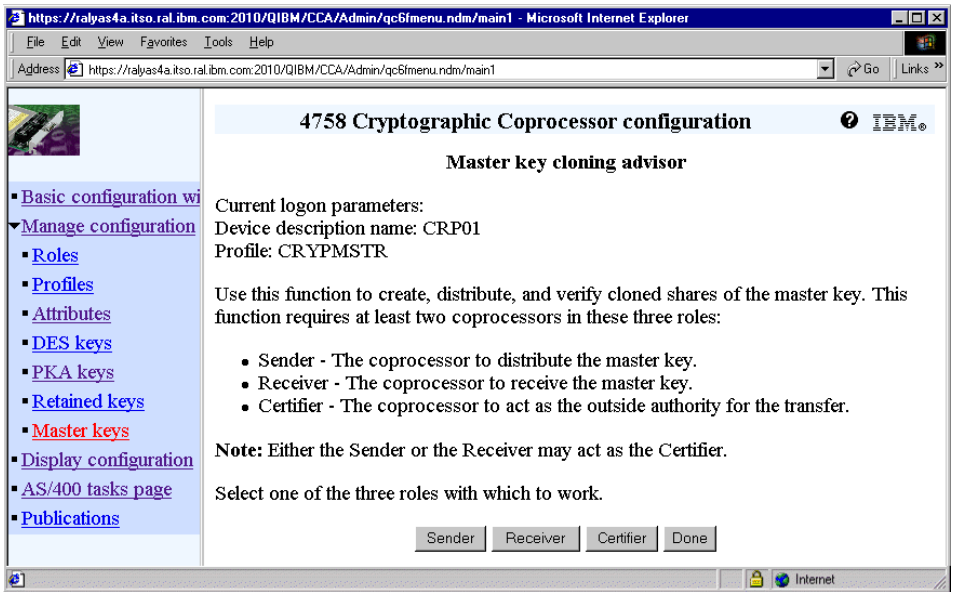


Figure 151. Master key cloning advisor window

17. Click **Done** to return to the Master key management window. Scroll down the window until you see the buttons shown in Figure 152.

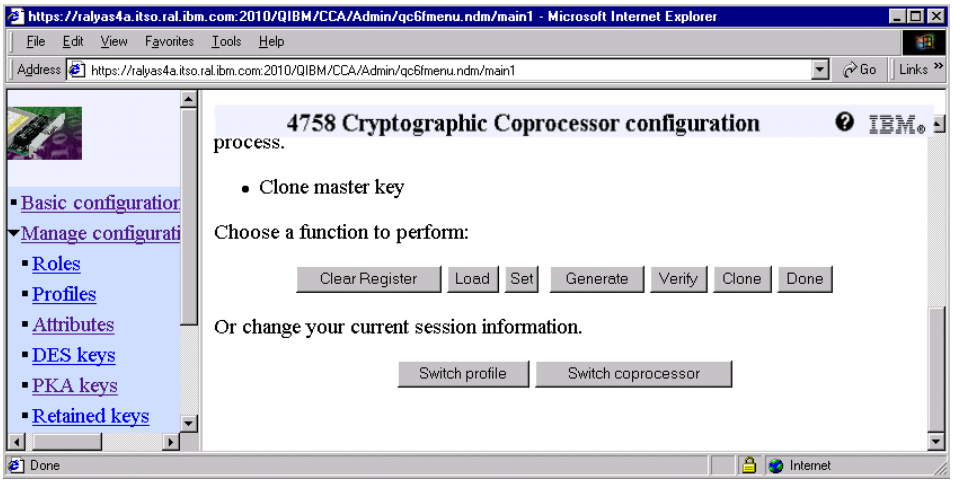


Figure 152. Master key management window

18. Click **Switch coprocessor** to start with the setup of the second coprocessor, the Receiver. The Allocate a device description window is displayed.

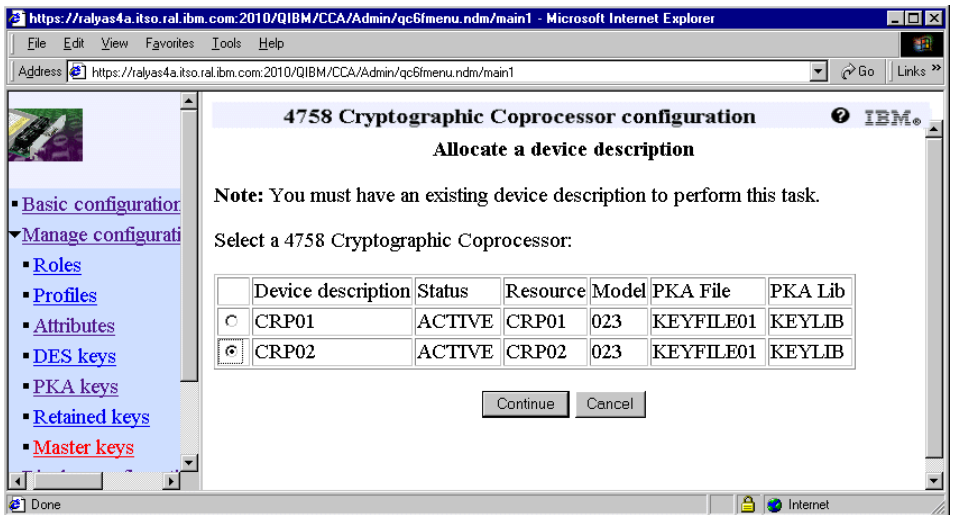


Figure 153. Allocate a device description window

Select **CRP02** or the device name where the master key has to be copied to.

19. Click **Continue**.

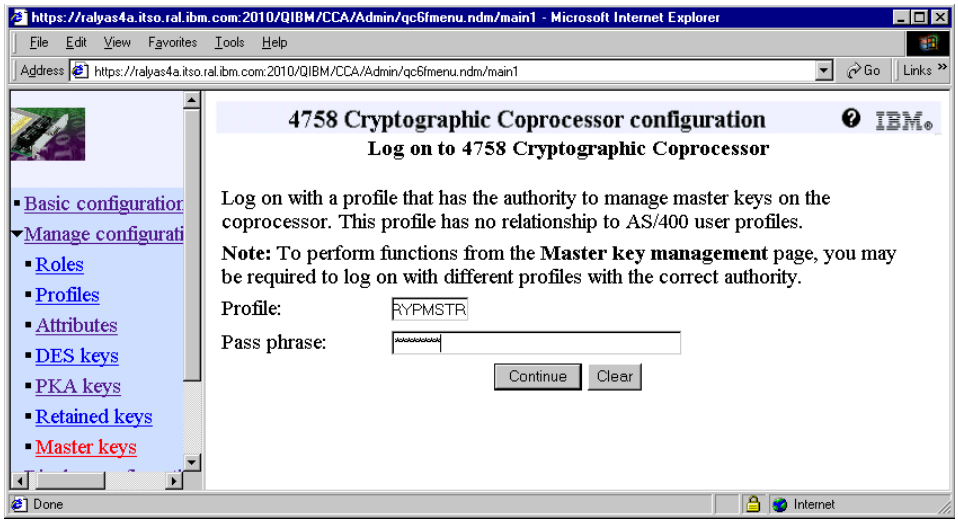


Figure 154. Log on to 4758 Cryptographic Coprocessor window

Enter CRYPMSTR as the profile and the profile's pass phrase. The profile and pass phrase information is case sensitive.

In this case the CRYPMSTR profile is also used, since it has all the authorities needed to perform master key cloning.

Every time you access or switch the coprocessor, the sign-on window for the specific 4758 Cryptographic Coprocessor will be displayed.

20. Click **Continue**.

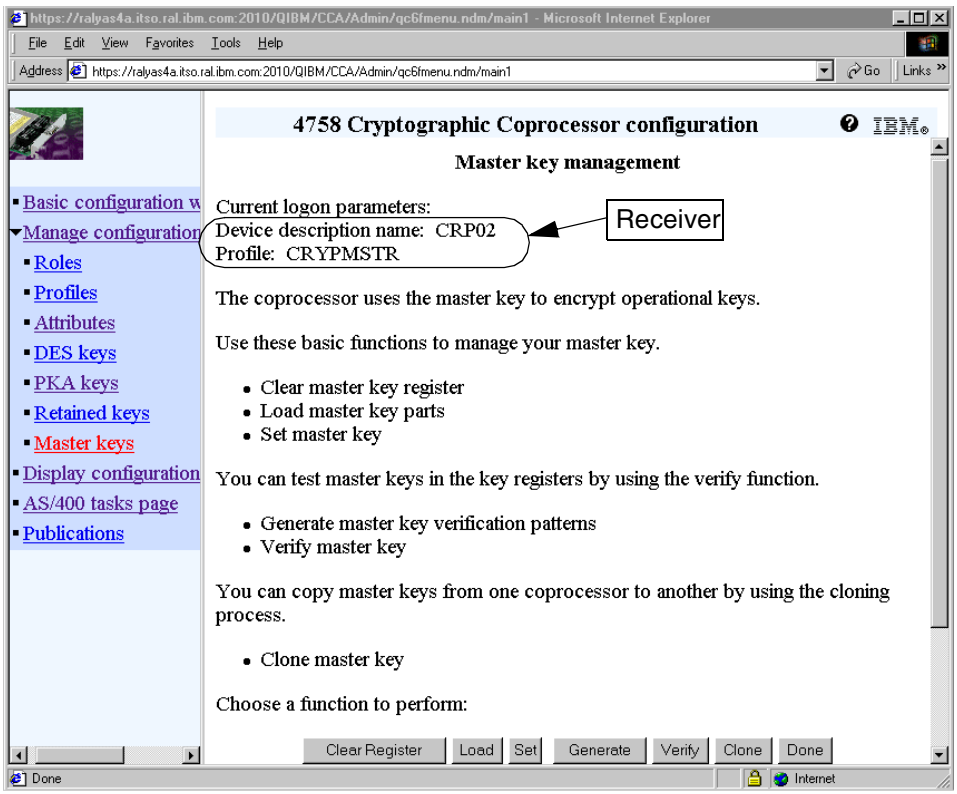


Figure 155. Master key management window

Verify that you work with the correct coprocessor. The device description name is displayed at the top of the Master key management window.

21. Click **Clone** to continue to the Master key cloning advisor for the Receiver coprocessor.

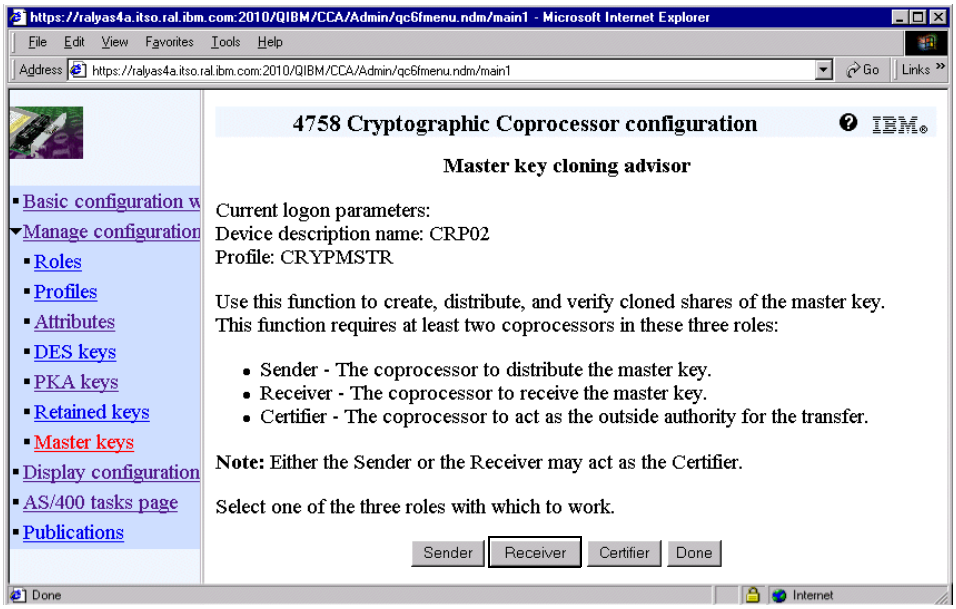


Figure 156. Master key cloning advisor window

22. Click **Receiver** to prepare the Receiver coprocessor.

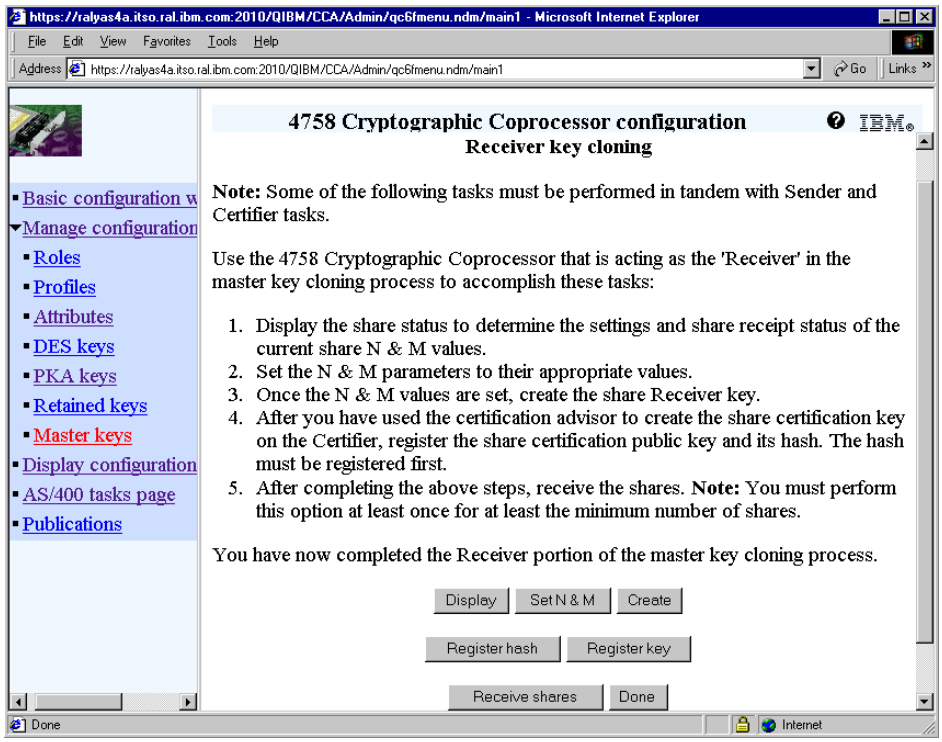


Figure 157. Receiver key cloning window

As already mentioned for the Sender part of the configuration, you do not need to perform the Display option, since there are no shares set at this time. Performing the Display would result in an error. Instead, continue with setting the number of N and M shares.

23. Click **Set N & M** to define the number of shares for N and M.

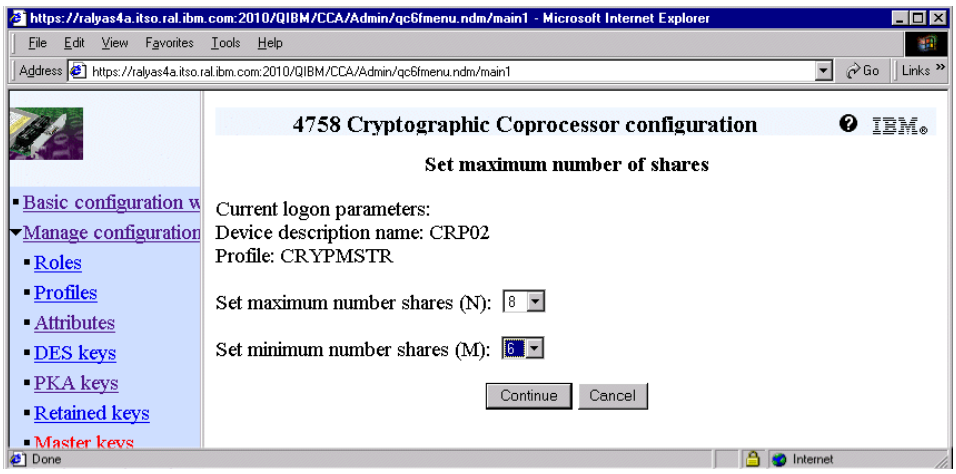


Figure 158. Set maximum numbers of shares window

The maximum and the minimum numbers of shares has to be set to the same values as they are on the Sender site.

Set the maximum and the minimum shares, as follows:

- Maximum number shares (N): 8
- Minimum number shares (M): 6

24. Click **Continue** to set the shares values.

    A confirmation message is displayed confirming that the share values have been set on the coprocessor.

25. Click **OK** on the confirmation message window to return to the Receiver key cloning window (the Receiver cloning window is shown in Figure 157 on page 244).

26. From the Receiver cloning window, click **Create** to create the private and public Receiver keys.

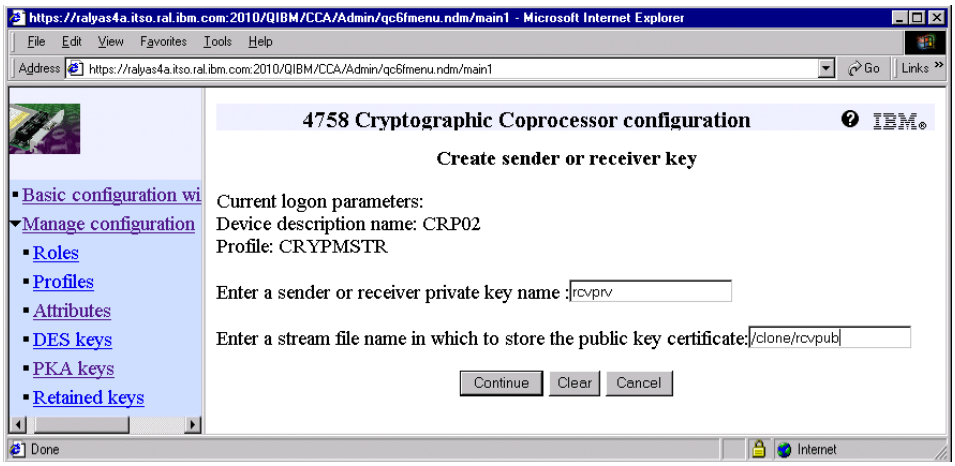


Figure 159. Create sender or receiver key window

Set the name for the Receiver's private key and indicate where the public key will be stored. Make sure the directory where the public key has to be stored exists prior to this action.

- Receiver private key name: `rcvprv`
- Stream file to store the Receiver's public key: `/clone/rcvpub`

27. Click **Continue**.

    A confirmation message shows up indicating that the key was created successfully.

28. Click **OK** to close the confirmation message window.

    You return to the Receiver key cloning window.

    The number of N & M shares have been set and the RSA key pair is created. That means the Receiver site setup is almost complete. But before continuing with the remaining Receiver configuration, the Certifier environment will be created and used to sign the Sender's and Receiver's public keys.

29. Click **Done** to return to the Master key cloning advisor window.

30. Click **Done** again on the Master key cloning advisor window to return to the Master key management window (see Figure 152 on page 240 for available options on the Master key management window).

31. Click **Switch coprocessor** to switch back to the first coprocessor (Sender site) to set up the Certifier. The Switch coprocessor button is located at the bottom of the window. Remember, the Certifier can be on a separate coprocessor, on the same coprocessor as the Sender, or on the same

coprocessor as the Receiver. In this scenario the Certifier is on the Sender's coprocessor.

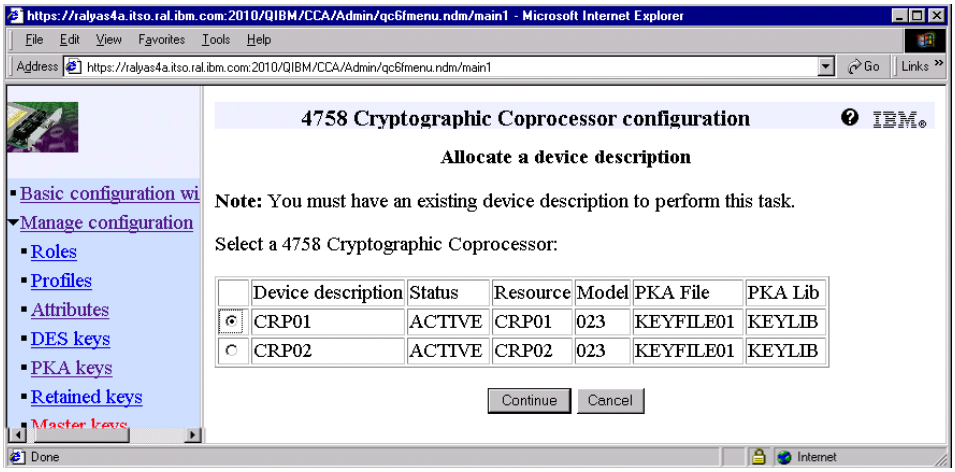


Figure 160. Allocate a device description window

32. Select **CRP01** and click **Continue**.

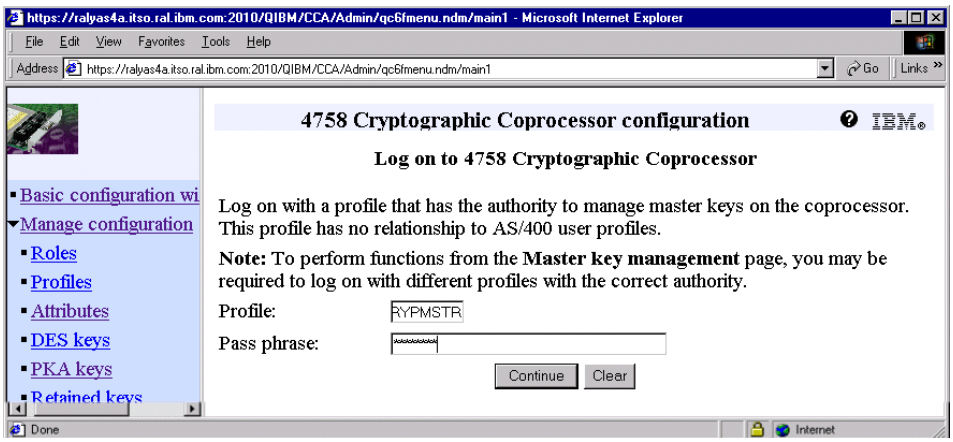


Figure 161. Log on to 4758 Cryptographic Coprocessor window

As already mentioned, the 4758 Cryptographic Coprocessor is a security device and therefore each access has to be granted.

Enter `CRYPMSTR` as the profile and the profile's pass phrase. The profile and pass phrase information is case sensitive.

33. Click **Continue**.

The Master key management window is displayed (see Figure 152 on page 240 for available options on the Master key management window).

34. Click **Clone** to display the Master key cloning advisor window, shown in Figure 162.

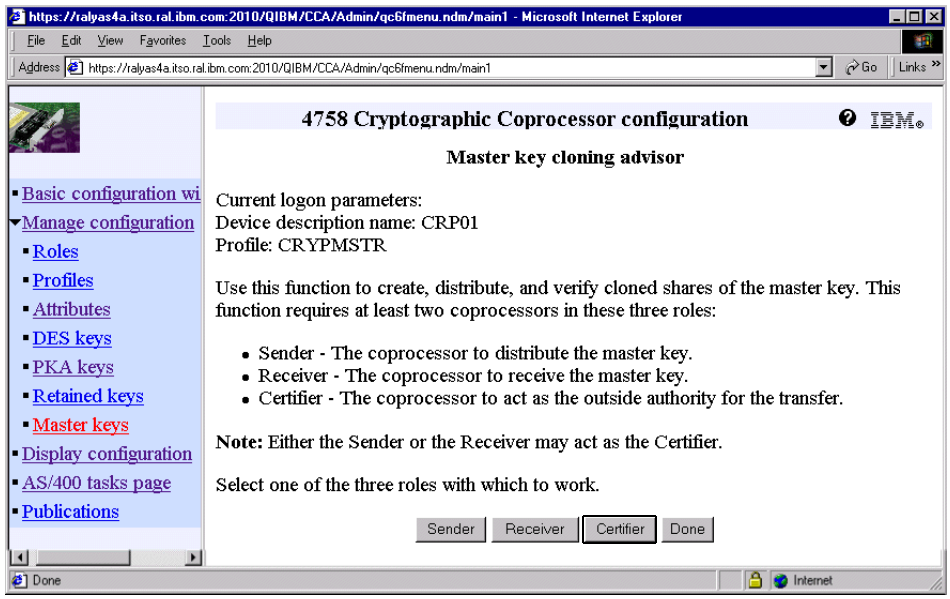


Figure 162. Master key cloning advisor window

The Certifier you are going to create will act as an outside Certificate Authority to the involved Sender and Receiver coprocessors, even when running on the Sender coprocessor. Both the Sender and the Receiver public keys need to be signed with the certifier private key. You can compare the certifier with an Internet Certificate Authority (CA), such as VeriSign, which signs digital certificates. In this case the Certifier signs just a public key.

35. Click **Certifier**.

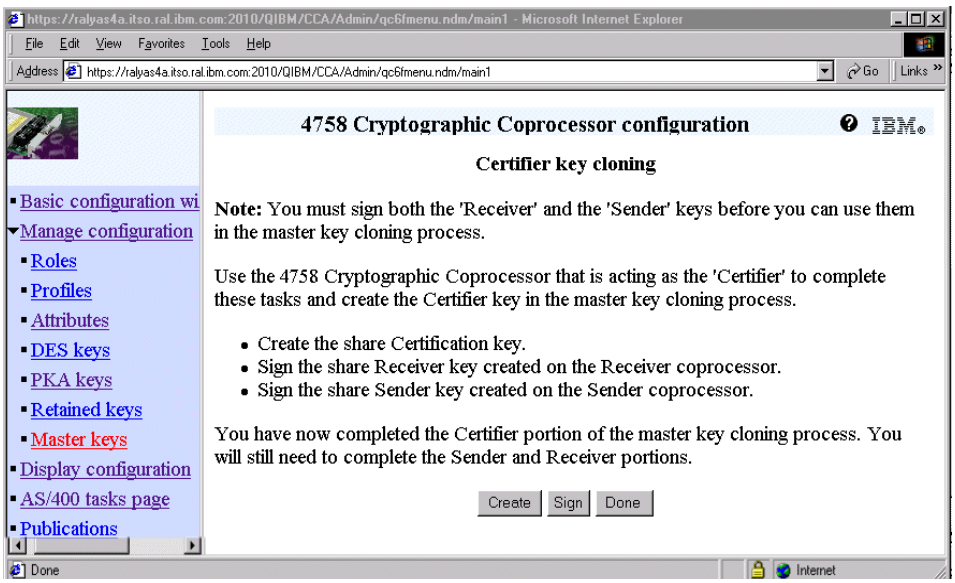


Figure 163. Certifier key cloning window

There are two tasks to perform: creating the RSA key pair for the Certifier and signing the Receiver and Sender public key.

36. Click **Create**.

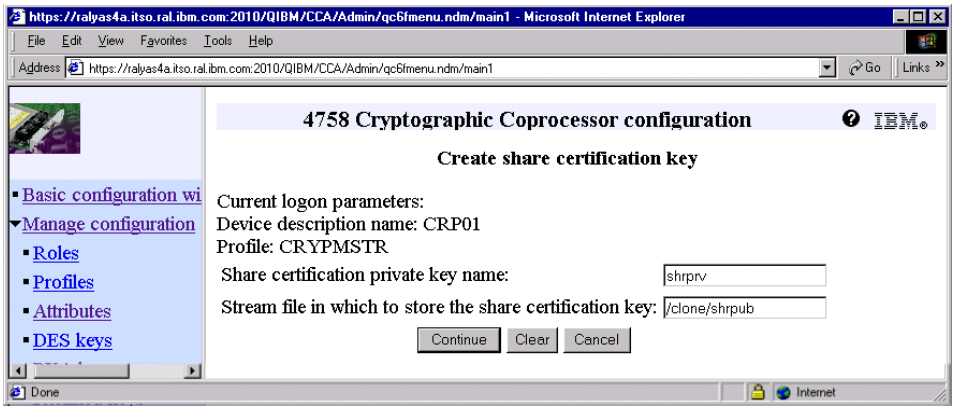


Figure 164. Create share certification key window

Set the name for the share certification private key and indicate where the public key will be stored. Make sure the directory where the public key has to be stored exists prior to this action.

- Share certification private key name: `shrprv`
- Stream file to store the share certification public key: `/clone/shrpub`

37. Click **Continue** to create the keys.

    A confirmation message shows up indicating that the key was created successfully.

38. Click **OK** to close the confirmation message window.

    You return to the Certifier key cloning window.

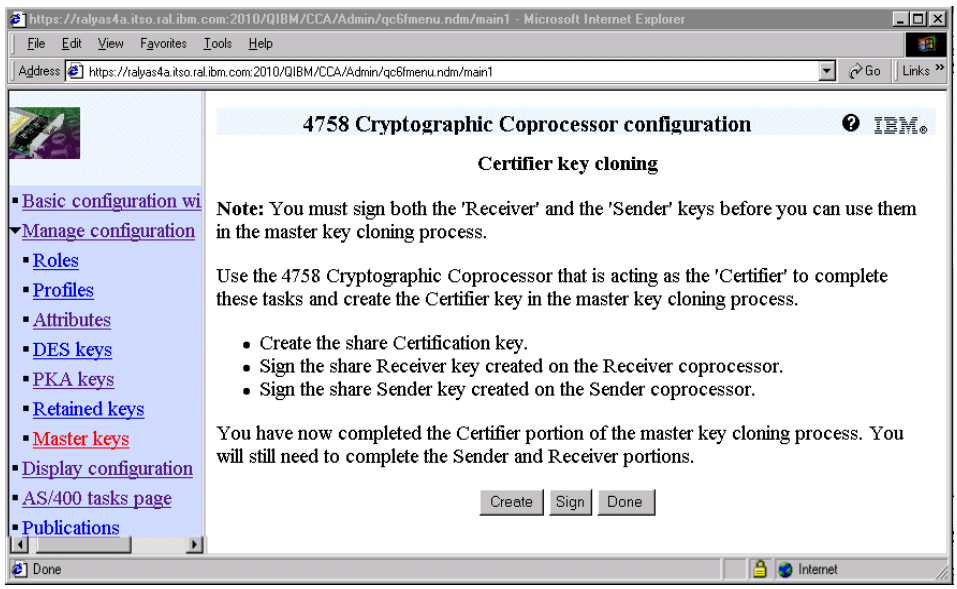


Figure 165. Certifier key cloning window

The public and the private keys are now created. In the following steps the share certification private key is used to sign the Sender and Receiver public key. The share certification public key is not used at this time. It will later on be used by the Sender to verify the signature of the Receiver's public key when creating the shares. It will also be used by the Receiver to verify the signature of the Sender's public key when receiving the shares.

39. Click **Sign** to sign the Sender and the Receiver keys.

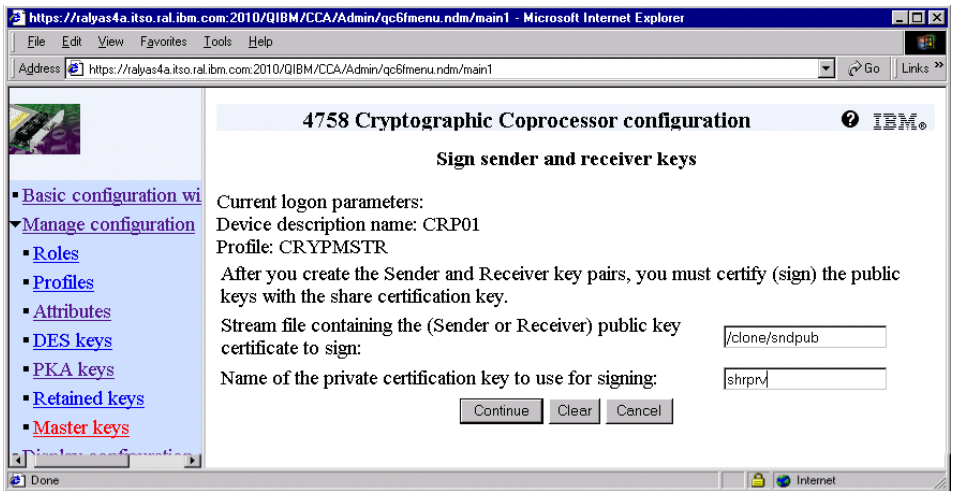


Figure 166. Sign sender and receiver keys window

Enter the directory and the name of the file containing the Sender's public key, and in the second field the name of the private certification key that was created in the previous step.

- Stream file containing Sender's public key: `/clone/sndpub`
- Name of private certification key: `shrprv`

40. Click **Continue** to sign the Sender's public key.

    A confirmation message will be shown stating that the unsigned Sender's public key has been replaced with a signed key.

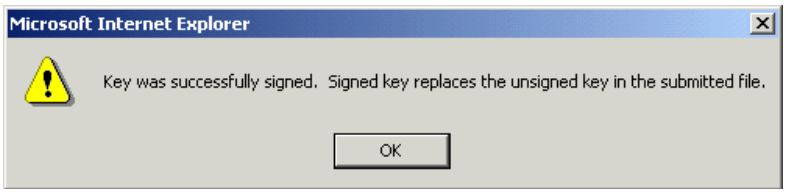


Figure 167. Key was successfully signed message

41. Click **OK** to return to the Certifier key cloning window.

42. Click **Sign** again to sign the Receiver's public key at this time.

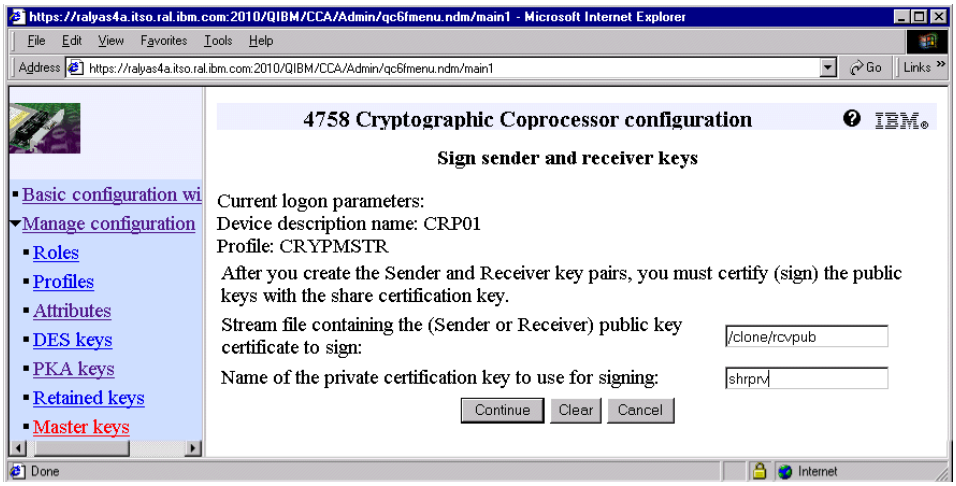


Figure 168. Sign sender and receiver keys window

Enter the directory and the name of the file containing the Receiver's public key, and in the second field the name of the private certification key that was created in the previous step.

- Stream file containing Receiver's public key: `/clone/rcvpub`
- Name of private certification key: `shrprv`

43. Click **Continue** to sign the Receiver's public key.

    A confirmation message will be shown stating that the unsigned Receiver's public key has been replaced with a signed key.

44. Click **OK** to return to the Certifier key cloning window.

45. Click **Done** to return to the Master key cloning advisor window.

The Certifier related tasks in the master key cloning process have been completed. During the next steps, the Certifier's share certification public key and its hash will be registered.

Since both the Sender and Certifier reside on the first coprocessor, it is not necessary to switch coprocessors at this time.

46. Click **Sender** on the Master key cloning advisor window (Refer to Figure 147 on page 236 to see the Master key cloning advisor window).

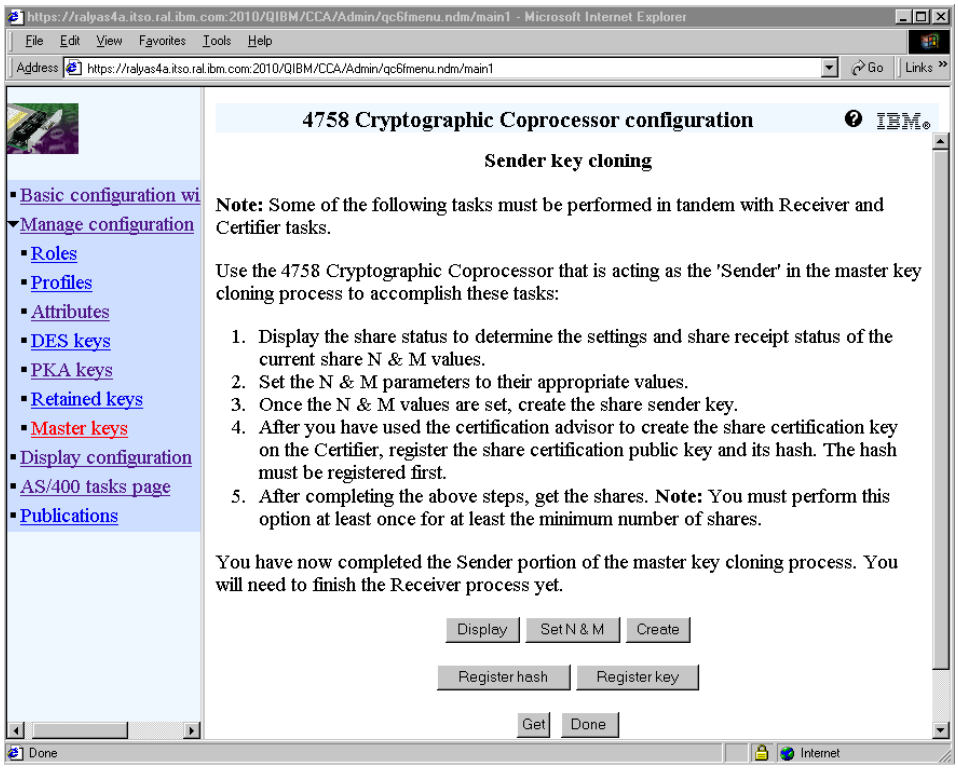


Figure 169. Sender key cloning window

First the hash of the Certifier's public share certification key will be registered in the Sender's coprocessor.

47. Click **Register hash**.

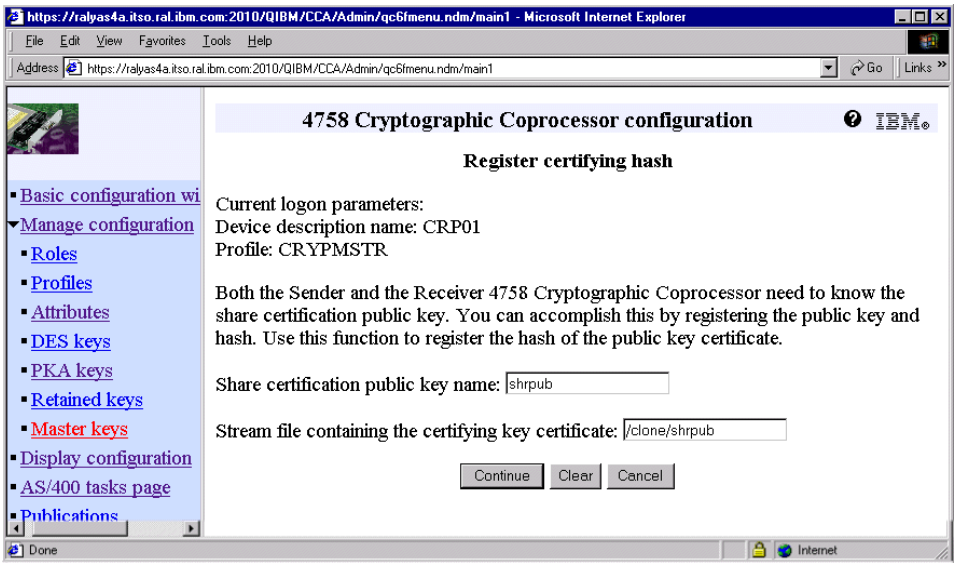


Figure 170. Register certifying hash window

Enter the name for the shared public key and the name of the file containing the public key of the Certifier.

- Share certification public key name: `shrpub`
- Stream file containing the certifying key: `/clone/shrpub`

48. Click **Continue** to register the hash.

    A confirmation message is displayed.

Figure 171. Hash registration message

49. Click **OK** to return to the Sender key cloning window.

50. Click **Register key** on the Sender key cloning window as shown in Figure 169 on page 253.

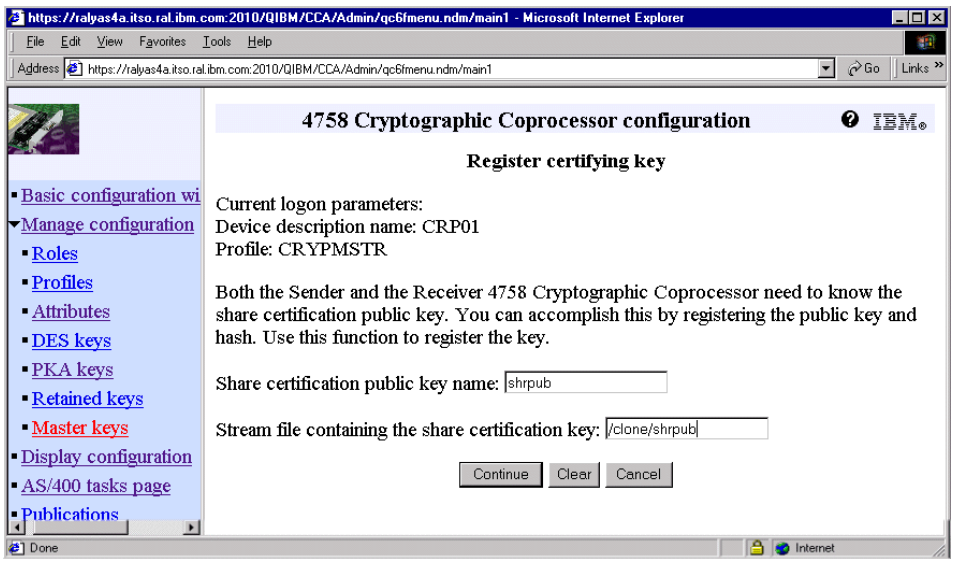


Figure 172. Register certifying key window

The key you are going to register is the public key of the Certifier. The parameter values are the same as for the hash registration.

- Share certification public key name: `shrpub`
- Stream file containing the certifying key: `/clone/shrpub`

51. Click **Continue** to register the key.

    A confirmation message will be displayed.

Figure 173. Key registration message

52. Click **OK** to return to the Sender key cloning window.

So far you completed the Certifier tasks and prepared the Sender site to create the shares. The Certifier's share certification key (public key) and its hash still still to be registered on the Receiver site. However, to save some steps, you continue with Sender site first and create all shares before switching over to the Receiver's coprocessor.

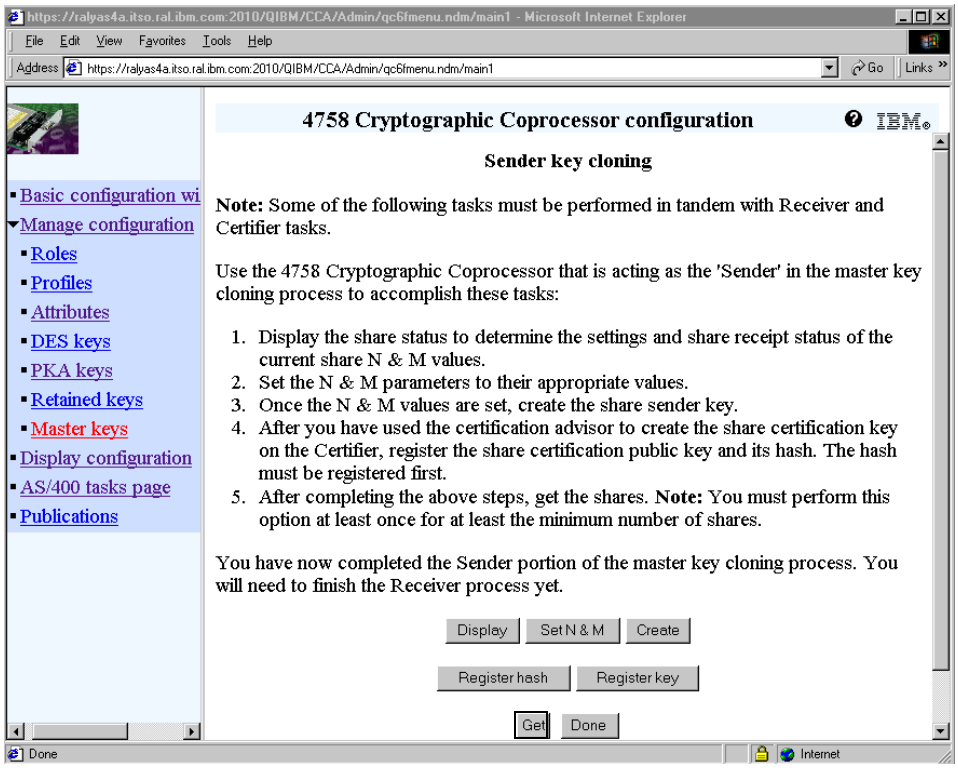


Figure 174. Sender key cloning window

In the next step you create the first share. In total you need to create eight shares as specified in the maximum number of shares (N).

53. Click **Get** to get the first share.

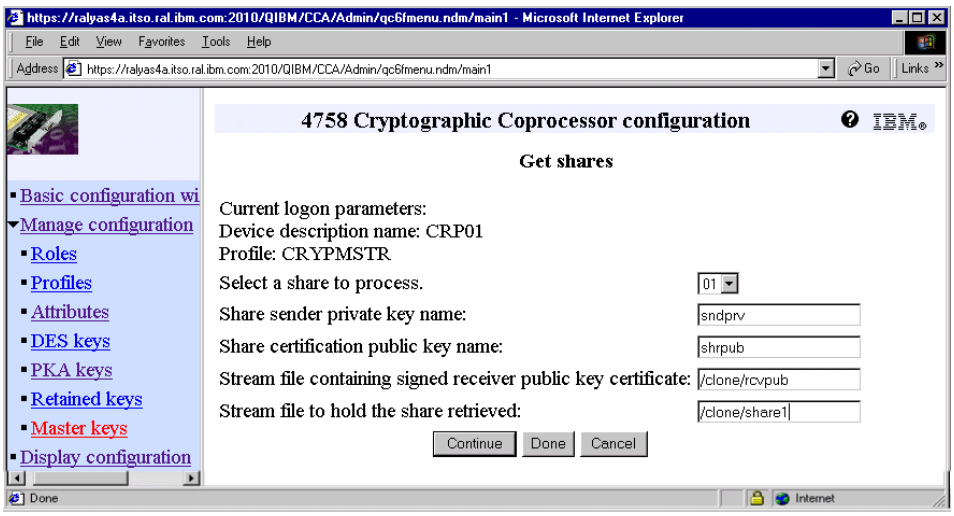


Figure 175. Get shares window

54. Enter the requested parameters as specified below.

    The Get shares step has to performed as many times as the number specified for N shares. Each time you repeat this step you need to increment the number of the share to process as well as the number in the file name that will hold the share.

    - Select a share to process: `01`
    - Share Sender private key name: `sndprv`
    - Share certification public key name: `shrpub`
    - Stream file containing signed Receiver public key: `/clone/rcvpub`
    - Stream file to hold the share retrieved: `/clone/share1`

55. Click **Continue** to retrieve the first share and store it in the specified stream file.

    A message is displayed confirming that the Get share completed.

> **Note**
>
> If you try to get (retrieve) more shares than specified by the maximum number of shares (N), a message is displayed with the following text:
>
> `The Master Key Share Index is not valid.`

56. Click **OK** to continue.

57. If this is the last share to be retrieved, click **Done**, otherwise repeat steps 54 and 56 until all shares are obtained for this coprocessor. Each cycle increment the share to be processed, for example `02, 03, 04`. You also need to change the name of the stream file that will hold the retrieved share, for example, `/clone/share2,/clone/share3`, and so forth. In this cloning scenario the maximum number of shares (N) is set to 8, so you have to perform the step 8 times.

    When you clicked Done you will return to the Sender key cloning window.

58. Click **Done** to return to the Master key cloning advisor window.

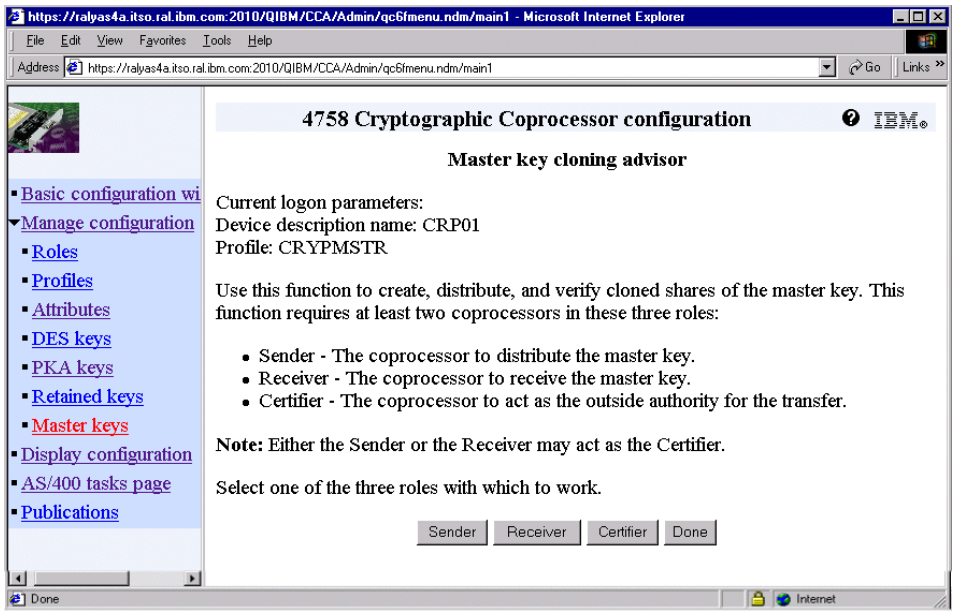


Figure 176. Master key cloning advisor window

59. Click **Done** to return to the Master key management window.

60. Scroll down the Master key management window and click **Switch coprocessor** to switch over to the Receiver coprocessor.

The Certifier and Sender tasks are now completed. You will continue with performing the remaining configuration tasks on the Receiver site and then receive the shares.

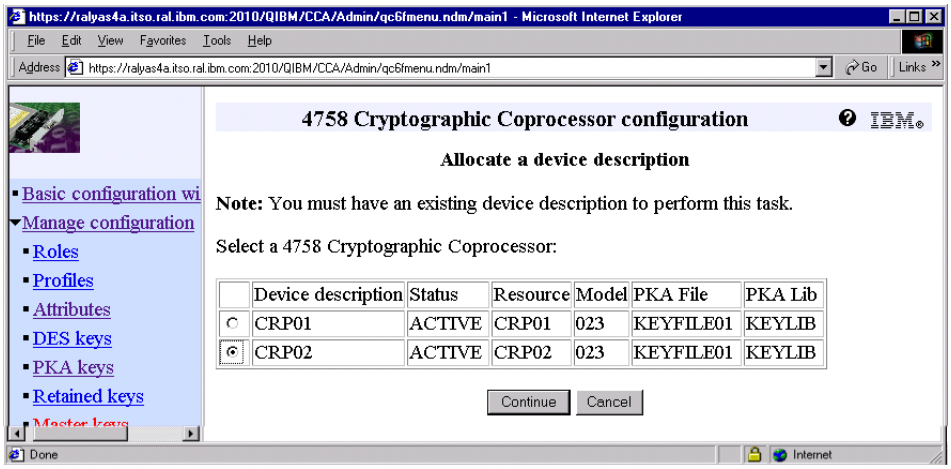


Figure 177. Allocate a device description window

61. Select coprocessor **CRP02** and click **Continue**. All the remaining steps of the master key cloning process will be performed on the Receiver coprocessor (CRP02).

    The Log on to 4758 Cryptographic Coprocessor window appears.

62. Enter `CRYPMSTR` as the profile and the profile's pass phrase. The profile and pass phrase information is case sensitive. For help on where to enter the profile and pass phrase, refer to Figure 145 on page 234, which shows the Log on to 4758 Cryptographic Coprocessor window.

63. Click **Continue** to complete the logon.

    The Master key management window is displayed.

64. Verify that you are working with the correct coprocessor. In this scenario it is the device CRP02, the Receiver.

65. Click **Clone** in the Master key management window.

    The Master key cloning advisor window is displayed.

66. Click **Receiver** to display the Receiver key cloning window.

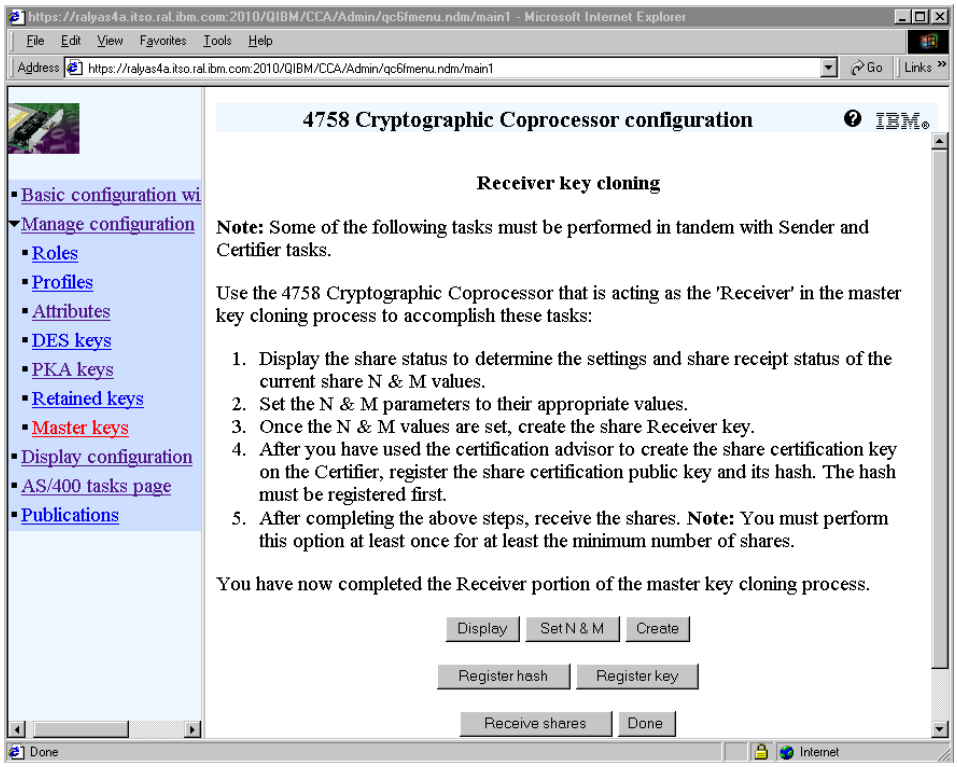


Figure 178. Receiver key cloning window

Now you finish the remaining setup steps for the Receiver coprocessor. This includes registering the Certifier's public or certification key and its hash.

67. Click **Register hash**.

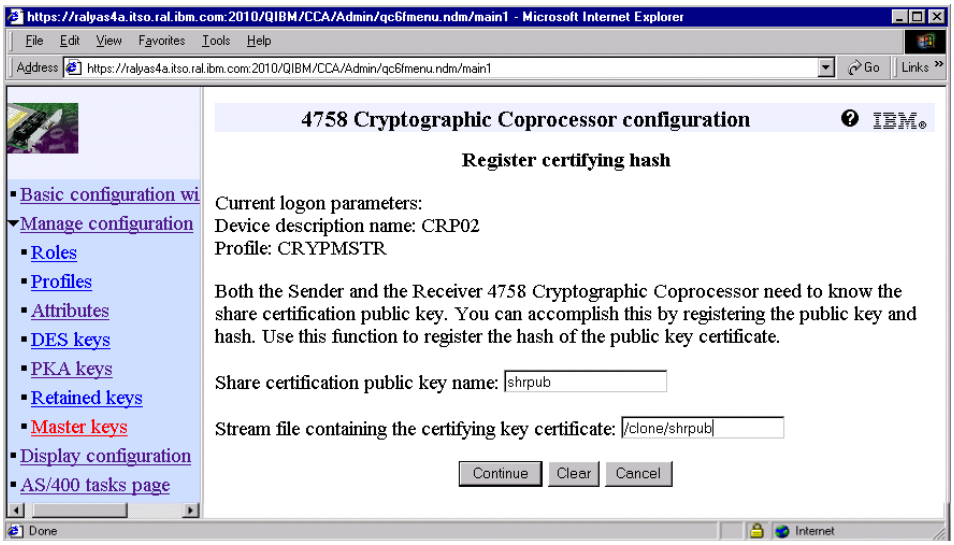


Figure 179. Register certifying hash window

Enter the name for the shared public key and the name of the file containing the public key of the Certifier.

- Share certification public key name: `shrpub`
- Stream file containing the certifying key: `/clone/shrpub`

68. Click **Continue** to register the hash.

    A confirmation message will be displayed. See Figure 180.

Figure 180. Hash registration message

69. Click **OK** to return to the Receiver key cloning window.

70. Click **Register key** on the Receiver key cloning window as shown in Figure 178.

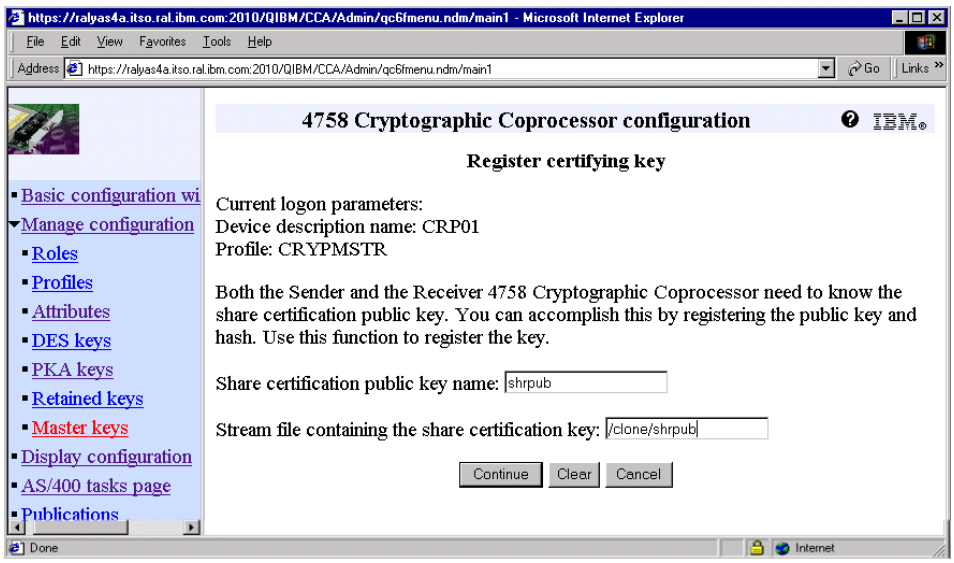


Figure 181. Register certifying key window

The key you are going to register is the public key of the Certifier. The parameter values are the same as for the hash registration.

- Share certification public key name: `shrpub`
- Stream file containing the certifying key: `/clone/shrpub`

71. Click **Continue** to register the key.

    A confirmation message will be displayed. See Figure 182.

Figure 182. Key registration message

72. Click **OK** to return to the Receiver key cloning window.

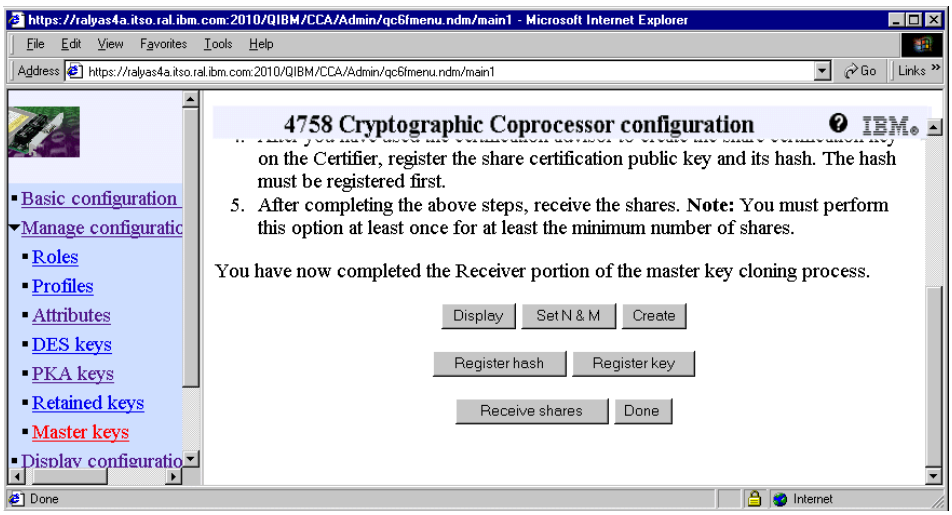


Figure 183. Receiver cloning window

73. Click **Receive shares**.

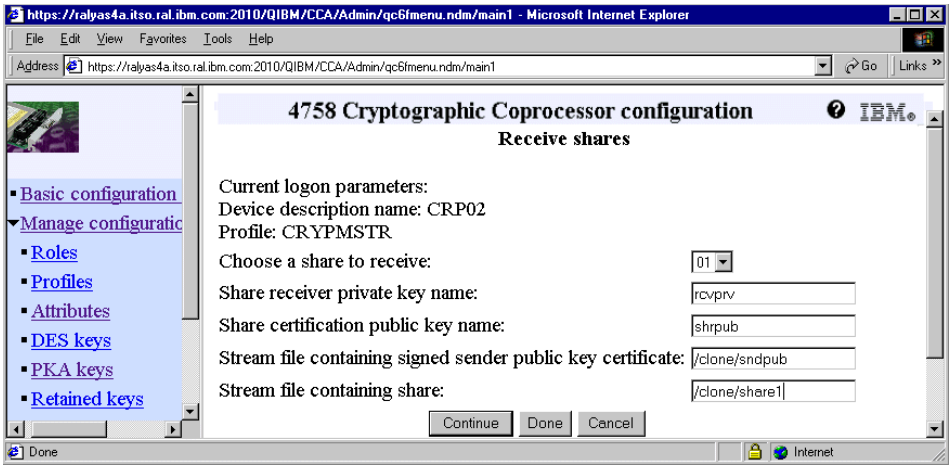


Figure 184. Receive shares window

74. Enter the requested parameter as specified below.

Finally you get to the point to receive the master key parts in to the Receiver coprocessor. Remember, you retrieved eight shares because it was specified for the maximum number of shares (N), but you need to receive only 6 shares as specified for the minimum number of shares (M). It actually does not matter which six out of the eight shares you pick, but

we recommend that you use the first six shares. If you used an enhanced security model that split the authorization into multiple profiles, you need to log on to the coprocessor with each of the profiles and receive the shares each profile is authorized to. In this scenario the cloning is performed by the CRYPMSTR profile without any special setup. So you can receive all shares with the same profile. Enter the parameter as follows:

- Choose a share to receive: `01`
- Share Receiver private key name: `rcvprv`
- Share certification public key name: `shrpub`
- Stream file containing signed Sender public key: `/clone/sndpub`
- Stream file containing share: `/clone/share1`

Increment the number of the share to be received and the name of the stream file containing the share each time you repeat this step.

75. Click **Continue** to receive the share.

    A confirmation message is displayed. See Figure 185.

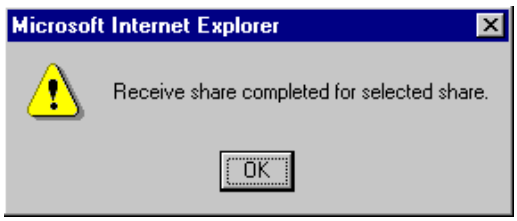


Figure 185. Receive share confirmation message

When the number of shares that are needed to rebuild the master key are received, you will receive the following message:

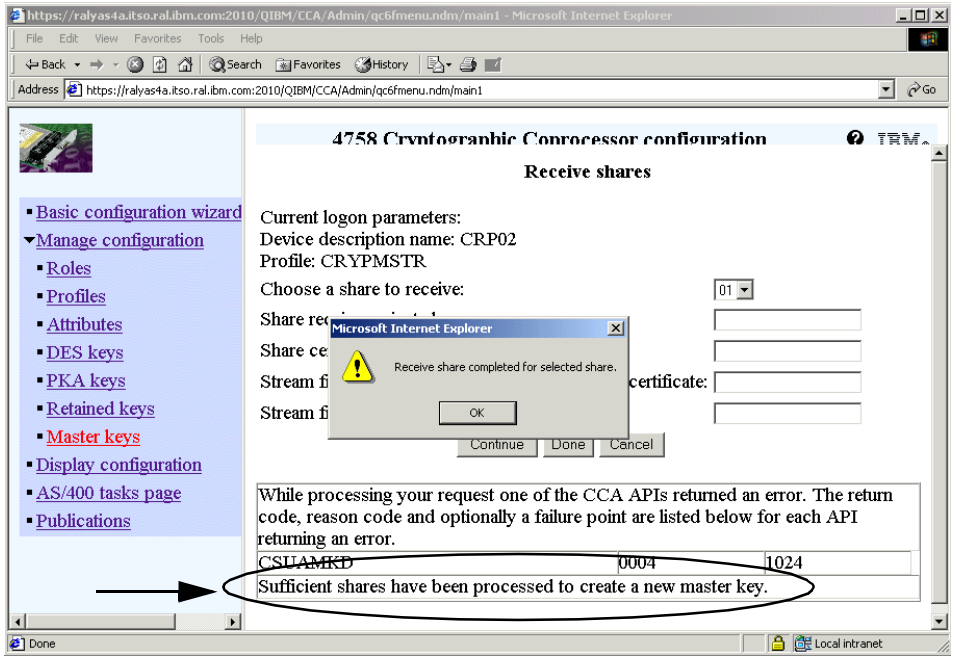


Figure 186. Sufficient number of shares received window

Another error you might get is when you try to receive more shares than specified for the minimum number of shares (M). The error message shown in this case is:

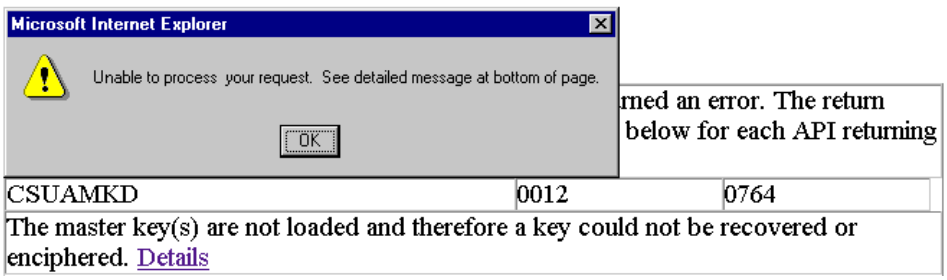


Figure 187. Error message when trying to receive more shares than allowed

76. Click **OK** to continue.

77. If this is the last share to be received, click **Done**. Otherwise, repeat steps 74 and 76 until the minimum number of shares are received for this coprocessor. Each cycle increments the share to be received, for example `02, 03, 04`. You also need to change the name of the stream file that

contains the share, for example, `/clone/share2`, `/clone/share3`, and so forth. In this cloning scenario the minimum number of shares (M) is set to 6, so you have to perform the step six times.

After you received the shares and clicked **Done**, you return to the Master key cloning advisor window.

78. Click **Done** again to return to the Master key management window.

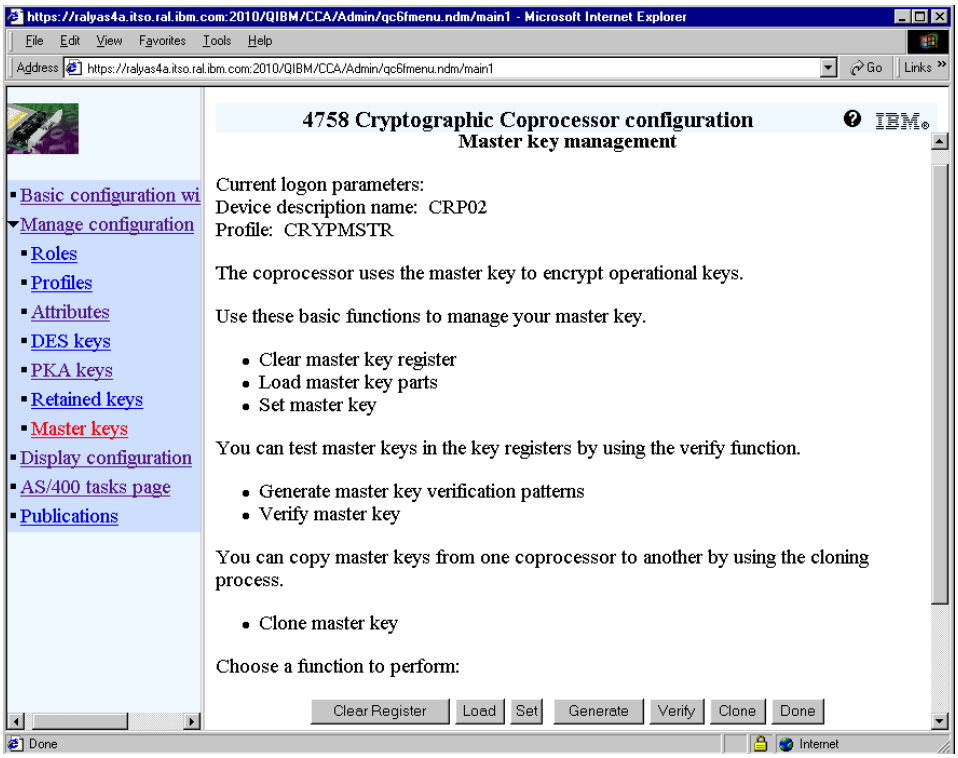


Figure 188. Master key management window

To summarize, all Sender and Certifier tasks are completed. All shares that are required to rebuild the master key on the Receiver coprocessor are received. The master key is reassembled and stored in the New Master Key register. In the next steps, you will activate the new master key on the Receiver coprocessor.

79. Click **Set** on the Master key management window.

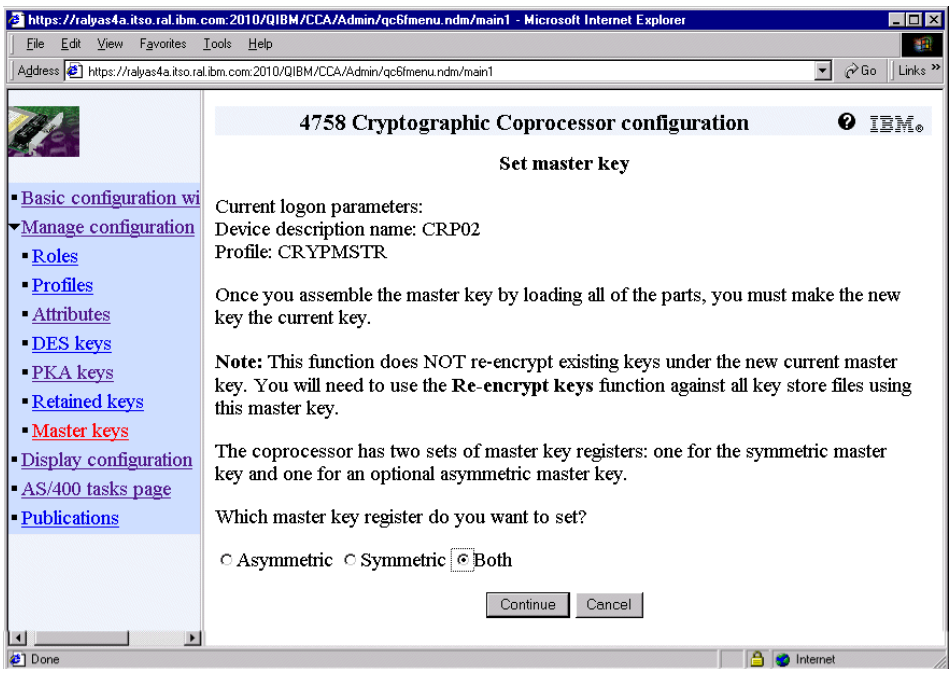


Figure 189. Set master key window

Setting the master key copies the master key from the New Master Key register to the Current Master Key register. Note the information in the middle of the right pane shown in Figure 189. It says that setting the master key does not re-encrypt the keys stored in key store files outside the coprocessor. Since we cloned the master key from another coprocessor and use the key store files that contain keys encrypted by the cloned master key, you do not need to re-encrypt the keys. This is true if the Receiver coprocessor is a new one and was not used before on this system to protect keys with the old master key.

To make sure that the master key for asymmetric and symmetric keys is the same, you have to select both.

Note

Verify again that you selected the correct coprocessor. In this scenario it is CRP02.

80. Select **Both** and click **Continue** to make the new master key the current one.

If you performed all the master key cloning steps in the right sequence and entered the correct information as documented in this chapter, you will see the following completion messages. You need to click **OK** on the first message to see the second message.

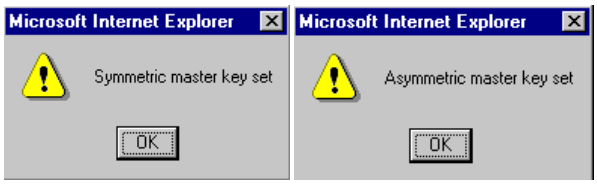


Figure 190. Setting the master key confirmation messages

Congratulations! You completed a long sequence of steps to clone the master key from the Sender coprocessor to the Receiver coprocessor. For the scenario described in this chapter, you still need to update the device assignment for the certificate or certificates that are assigned to the SSL applications you want to use load sharing for. Refer to 4.6, “Updating the 4758 Cryptographic Coprocessor device assignment” on page 218, for detailed information on how to perform this task.

4.9 Operating the 4758 Cryptographic Coprocessor

Once the 4758 Cryptographic Coprocessor is properly configured, there are almost no further actions required to use the coprocessor.

In order for an application to use the cryptographic coprocessor, the coprocessor has to be varied on. Use the `WRKCFGSTS *DEV *CRP` command to determine the configuration status of all configured cryptographic coprocessors on the system.

If you use the 4758 Cryptographic Coprocessor to perform SSL session establishment tasks and, for example, you update the device assignment for a certificate to add another coprocessor device for load sharing, you need to restart the application that uses the certificate. This action is required to allow the application, for example, an HTTP server, to use both coprocessors for load sharing.

You can determine the jobs using a coprocessor by also using the `WRKCFGSTS *DEV *CRP` command and look for the jobs associated with the individual coprocessor. Figure 191 shows two coprocessors, CRP01 and CRP02. The ADMIN job is associated only with CRP01, which means no load sharing is involved. Both coprocessors CRP01 and CRP02 show the same

application associated with them, which means the certificate that the HTTP server THOMASBSSL is using has both coprocessors assigned for load sharing.

```
                        Work with Configuration Status                AS4A
                                                          03/23/01  23:43:24
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status    13=Work with APPN status...

Opt  Description       Status                -------------Job--------------
     CRP01             ACTIVE                ADMIN       QTMHHTTP    007978
                       ACTIVE                THOMASBSSL  QTMHHTTP    008319
                       ACTIVE                THOMASBSSL  QTMHHTTP    008320
     CRP02             ACTIVE                THOMASBSSL  QTMHHTTP    008319
                       ACTIVE                THOMASBSSL  QTMHHTTP    008320




                                                                        Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 191. Configuration status of cryptographic coprocessors

4.10 Initializing the 4758 Cryptographic Coprocessor adapter

This section discusses a topic you should familiarize yourself before performing any of the documented steps. It is about re-initializing the 4758 Cryptographic Coprocessor. Under normal circumstances there is no need to re-initialize the coprocessor, except for such reasons as:

- The coprocessor has been sold and you want to delete all your configuration and retained keys before giving the adapter away. Keep in mind that on the iSeries server, the configuration remains in the coprocessor even when the card is physically removed from the system. This is not the case in some other implementations. Leaving the keys on the adapter could expose very sensitive information.
- You want to re-purpose one coprocessor, for example for load sharing, and do not need the current configuration anymore. Re-initializing the coprocessor allows you to start with a clean configuration.
- You forgot the pass phrases of the profiles allowed to change the coprocessor configuration. There is no other way than re-initializing the coprocessor via the hardware service manager.

- You accidently set up the profiles and their roles so that no profile can perform certain tasks anymore.

Important

The re-initialization process destroys all the data on the coprocessor adapter and resets the 4758 Cryptographic Coprocessor to its factory settings. All keys not backed up or exported will be removed, along with the master key if not cloned to another cryptographic coprocessor. This could cause serious problems for the company if all the keys are lost.

There are two different methods of re-initializing the 4758 Cryptographic Coprocessor:

- Using the graphical user interface (GUI) of the 4758 Cryptographic Coprocessor configuration utility.
- Using the hardware service manager of the System Service Tools (SST).

Both methods are described in the following section.

4.10.1 Using the GUI to re-initialize the coprocessor

The graphical user interface is one of the methods for re-initializing the coprocessor. This method requires that you can log on to the coprocessor with a profile that has enough authority to re-initialize the coprocessor. The system generated profiles when using the 4758 Cryptographic Coprocessor configuration wizard that have the proper level of authority are:

- Coprocessor configured for one profile: CRYPMSTR
- Coprocessor configured for three profiles: CRYPADMN

There is no way of using the GUI when you do not know the pass phrases of the profiles. If this is the case, you have to perform the re-initialization through the hardware service manager in SST.

This option deletes the entire configuration, such as keys, profiles, roles, and environment ID, but it does not clear the Function Control Vector (FCV). If you want to clear this value too, you have to do this before re-initializing the coprocessor by using the Clear FCV option on the Attribute Management window.

Perform the following steps to re-initialize the 4758 Cryptographic Coprocessor using the GUI:

1. Start the AS/400 Tasks page using a Web browser and enter the following URL:

   `http://servername:2001`

   Where *servername* represents your iSeries host name. The options available from the AS/400 Tasks page vary depending on the installed program products.

2. After signing on to the AS/400 Tasks page, click **4758 Cryptographic Coprocessor**.

3. Click **Start secure session** to launch the 4758 Cryptographic Coprocessor configuration window. Note that you do get this option if you started the AS/400 Tasks page over a secure connection (port 2010).

4. On the navigation bar, click **Manage configuration** to expand its options and click **Attributes**.

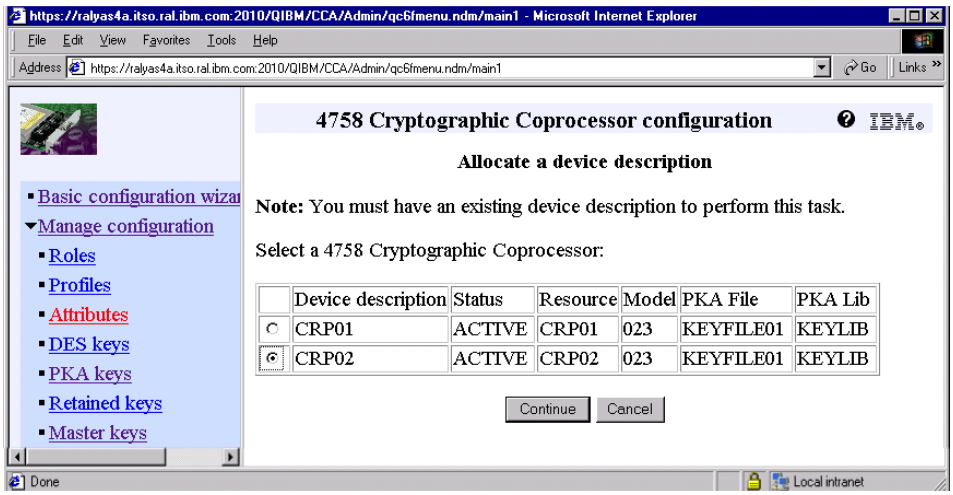


Figure 192. Allocate a device description window

   Note that the list contains only 4758 Cryptographic Coprocessors that are configured in the system, that is, when a device description exists. Coprocessor adapters that are physically installed in the system but not configured are not displayed here, but you can still re-initialize these coprocessors by using the hardware service manager in SST as described in 4.10.2, “Using System Service Tools to re-initialize coprocessor” on page 274.

5. Select the coprocessor you want to re-initialize and click **Continue**.

Important

If you have more than coprocessor installed, double-check that you selected the correct one. The consequences of re-initializing the wrong coprocessor could be fatal.

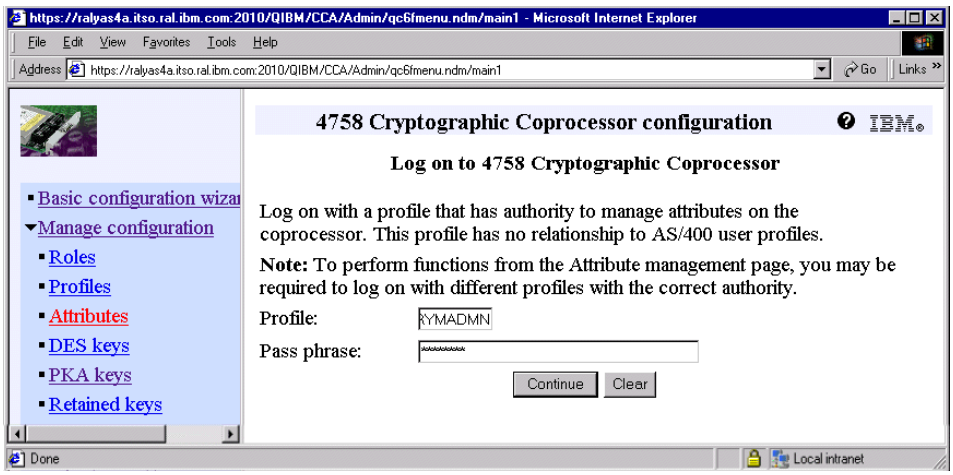


Figure 193. Log on to 4758 Cryptographic Coprocessor window

Log on to the coprocessor with a profile having sufficient authority to perform the re-initialization. For a list of default authority settings for the system-generated profiles and the associated hardware commands, refer to Appendix A, “4758 cryptographic coprocessor hardware commands” on page 403.

In this example, the coprocessor was initially configured to be operated by three profiles. Thus, the CRYPADMN profile is used.

6. Enter the profile and pass phrase and click **Continue**.

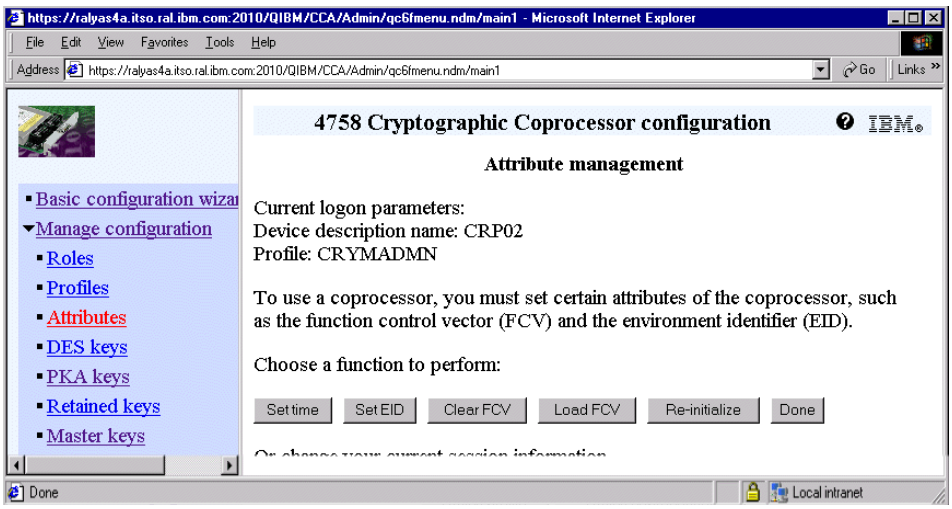


Figure 194. Attribute management window

7. Click **Re-initialize** in the Attribute management window.

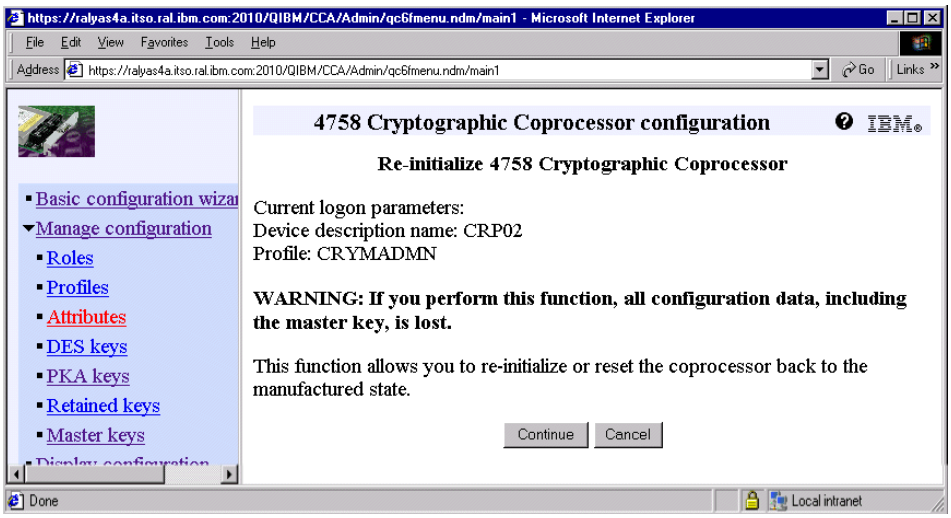


Figure 195. Re-initialize 4758 Cryptographic Coprocessor window

A warning message is displayed giving you the last chance to cancel the re-initialization. If you want to abort the re-initialization process, click **Cancel** now.

8. Click **Continue** to start the re-initialization process. All keys stored on this coprocessor including the master key will be deleted. After a short time the

following message window is displayed telling you that the 4758 Cryptographic Coprocessor has been re-initialized.

Figure 196. Re-initialization completion message

4.10.2 Using System Service Tools to re-initialize coprocessor

Using the hardware service manager provided in the System Service Tools (SST) allows you to re-initialize any of the installed 4758 Cryptographic Coprocessors. This is the only method to re-initialize a coprocessor when the pass phrase of a profile authorized to perform re-initialization via the GUI is not known.

Perform the following steps to re-initialize the 4758 Cryptographic Coprocessor using the hardware service manager:

1. Log on to a 5250 display using an OS/400 user profile that has authority to the `STRSST` system command and *SERVICE special authorities.
2. Enter the command `WRKHDWRSC TYPE(*CRP)` to display all 4758 Cryptographic Coprocessors that are installed on the system.

```
                    Work with Cryptographic Resources
                                                          System:   AS4A
Type options, press Enter.
  5=Work with configuration descriptions   7=Display resource detail

Opt  Resource        Type  Status              Text
     CMB03           2842  Operational         Combined function IOP
       CRPCTL01      4758  Operational         Cryptographic IOA
         CRP01       4758  Operational         Cryptographic Device
       CRPCTL02      4758  Operational         Cryptographic IOA
         CRP02       4758  Operational         Cryptographic Device




                                                                  Bottom
F3=Exit   F5=Refresh   F6=Print   F12=Cancel
```

Figure 197. Work with Cryptographic Resources menu

All installed coprocessors are displayed. This list does not necessarily match the number of configured coprocessors, that is, where a device description was created for. If you want to know whether a device description exists, you can use option 5 before the device entry to view the device descriptions.

Write down the cryptographic device you want to re-initialize.

3. Press F3 to exit from the cryptographic resources window.

   Before re-initializing a coprocessor using the hardware service manager, the coprocessor must be varied off.

4. Enter the command `WRKCFGSTS CFGTYPE(*DEV) CFGD(CRP01)` to check the configuration status of the coprocessor you want to re-initialize.

```
                        Work with Configuration Status                AS4A
                                                          03/24/01  16:26:55
Position to  . . . . .               Starting characters

Type options, press Enter.
  1=Vary on   2=Vary off   5=Work with job   8=Work with description
  9=Display mode status    13=Work with APPN status...

Opt  Description       Status                -------------Job--------------
 2   CRP01             ACTIVE



                                                                      Bottom
Parameters or command
===>
F3=Exit   F4=Prompt   F12=Cancel   F23=More options   F24=More keys
```

Figure 198. Cryptographic coprocessor configuration status

If the coprocessor shows active jobs, end the jobs first in an orderly manner before varying off the device.

5. If not already varied off, select option `2` to vary off the cryptographic coprocessor.

6. Enter the command `STRSST` from the OS/400 command line.

```
                    Start Service Tools (STRSST) Sign On

                                                  SYSTEM: AS4A

 Type choice, press Enter.

   Service tools user . . . . . QSECOFR
   Service tools password . . .



 F3=Exit      F9=Change Password         F12=Cancel
```

Figure 199. Start Service Tools Sign On window

With OS/400 V5R1 you need to sign on to SST with a service tools user. This is not an OS/400 user profile. One of the standard user IDs is QSECOFR.

7. Enter a service tools user, such as `QSECOFR` and its password and press Enter to sign on to SST.

Attention

Incorrect use of the System Service Tools can cause damage on the system or on data. Do not perform any other SST tasks unless you are directed to do so by an IBM service representative. If you are not familiar with SST or do not want to work with it, call IBM and have a service representative perform the necessary steps.

```
                        System Service Tools (SST)

 Select one of the following:

      1. Start a service tool
      2. Work with active service tools
      3. Work with disk units
      4. Work with diskette data recovery
      5. Work with system partitions
      6. Work with system capacity

Selection
    1
 F3=Exit         F10=Command entry         F12=Cancel
```

Figure 200. System Service Tools (SST) main menu

8. Select option 1 (Start a service tool) and press Enter.

```
                         Start a Service Tool

 Warning: Incorrect use of this service tool can cause damage
 to data in this system.  Contact your service representative
 for assistance.

 Select one of the following:

      1. Product activity log
      2. Trace Licensed Internal Code
      3. Work with communications trace
      4. Display/Alter/Dump
      5. Licensed Internal Code log
      6. Main storage dump manager
      7. Hardware service manager

Selection
     7
F3=Exit         F12=Cancel         F16=SST menu
```

Figure 201. Start a Service Tool

9. Select option 7 (Hardware service manager) and press Enter.

```
                  Hardware Service Manager

 Attention:  This utility is provided for service representative use only.

    System unit . . . . . . . :   9406-270 10-5F68M
    Release . . . . . . . . . :   V5R1M0 (1)

 Select one of the following:

    1. Packaging hardware resources (systems, frames, cards,...)
    2. Logical hardware resources (buses, IOPs, controllers,...)
    3. Locate resource by resource name
    4. Failed and non-reporting hardware resources
    5. System power control network (SPCN)
    6. Work with service action log
    7. Display label location work sheet
    8. Device Concurrent Maintenance

Selection
     2
 F3=Exit      F6=Print configuration      F9=Display card gap information
 F10=Display resources requiring attention      F12=Cancel
```

Figure 202. Hardware Service Manager menu

10. Select option 3 (Locate resource by resource name).

```
                    Locate Resource By Resource Name


Type resource name to be located, press Enter.

Resource name . . . . . .  crp01




F3=Exit      F9=Unmapped Resource Names      F12=Cancel
```

Figure 203. Locate Resource by Resource Name prompt

11. Enter the hardware resource name as determined in step 2. on page 274 and press Enter. In this example it is the resource `CRP01`. Double-check that this is the coprocessor you want to re-initialize.

```
                          Logical Hardware Resources

Type options, press Enter.
  2=Change detail    4=Remove    5=Display detail    6=I/O Debug
  7=Verify           8=Associated packaging resource(s)

                                                          Resource
Opt Description                    Type-Model  Status     Name
 6  Cryptography Device             4758-023   Operational CRP01




F3=Exit      F5=Refresh      F6=Print      F9=Failed resources
F10=Non-reporting resources  F11=Display serial/part numbers    F12=Cancel
 CRP01      located successfully.
```

Figure 204. Logical Hardware Resource list

12. Select option `6` (I/O Debug) end press Enter.

    If you receive the error message `A service session could not be started for this device`, there is still a device description varied on using this hardware resource. Use the command `VRYCFG CFGOBJ(CRP01) CFGTYPE(*DEV) STATUS(*OFF)` from a command line to vary off the device and try this step again.

```
                    Select Cryptography Debug Function

 Select one of the following:

      1. Reinitialize Flash Memory
      2. Select IOP Debug Function





Selection
   1
F3=Exit      F12=Cancel
```

Figure 205. Select Cryptographic Debug Function menu

13. Select option `1` (Reinitialize Flash Memory) and press Enter to re-initialize the flash memory on the selected coprocessor adapter.

```
                    Reinitialize Flash Memory Function

 DANGER:

   Performing this initialization of the flash memory on the cryptography
   device will cause ALL key information stored on the device to be
   DESTROYED.  This will cause all data encrypted using this device to be
   rendered unusable.

 WARNING:
    Performing this initialization of the flash memory on the cryptography
   device will take an estimated 10 minutes.


 Press Enter to proceed.
F3=Exit      F12=Cancel
```

Figure 206. Reinitialize Flash Memory Function warning message

A warning message is displayed giving you the last chance to cancel the re-initialization. If you want to abort the re-initialization process press F3 or F12 now.

14. Press Enter to start the re-initialization process. All keys stored on this coprocessor including the master key will be deleted. The following message is displayed providing information about the re-initialization progress:

```
                    Reinitialize Flash Memory Status


Flash memory reinitialization in progress...


 Estimated time: 10.0 minutes

 Elapsed time:    1.0 minutes
```

Figure 207. Reinitialize Flash Memory Status window

After the re-initialization of the 4758 Cryptographic Coprocessor has completed, the following message is shown at the bottom of the window:

```
Re initialization of cryptography device was successful
```

When we wrote this chapter, the average time needed to re-initialize the coprocessor was approximately three minutes.

4.11 Available APIs for the 4758 Cryptographic Coprocessor

Prior to OS/400 V5R1, the 4758 Cryptographic Coprocessor could only be used when writing your own applications. The applications used application programming interfaces (APIs) to access the coprocessor and perform functions on it. This included applications used to perform the coprocessor setup and administration. With V5R1 you can set up and manage the coprocessor by using the 4758 Cryptographic Coprocessor configuration utility available from the AS/400 Tasks page.

When you want to use the coprocessor only for performing SSL session establishment tasks, the 4758 Cryptographic Coprocessor configuration utility is the only tool required to set up the coprocessor.

If you plan to use other coprocessor functions, such as encryption, PIN processing services, or Message Authentication Code generation, you have to buy existing programs from software vendors or write your own applications using the available APIs.

Many example programs written in C and RPG are available to show you how to use the cryptographic coprocessor APIs. All the APIs are available via the OS/400 Common Cryptographic Architecture Cryptographic Service Provider (5722-SS1 Option 35) license program.

For API coding examples and information refer to:

- Document *4758 PCI Cryptographic Coprocessor for iSeries* found in the iSeries Information Center by clicking **Security->4758 PCI Cryptographic Coprocessor for iSeries**
- *IBM 4758 PCI Cryptographic Coprocessor CCA Basic Services Reference and Guide* found at
  `http://www.ibm.com/security/cryptocards/html/library.shtml`
- PCI Cryptographic Coprocessor information in iSeries Technical Studio at
  `http://www.as400.ibm.com/tstudio/tech_ref/security/crypto/index.htm`

4.12 Backup considerations

Before going into the details about the backup possibilities with the 4758 Cryptographic Coprocessor, you need to understand the following. The 4758 Cryptographic Coprocessor is a security device used to provide a high level of security and protection for sensitive data. This is accomplished by storing the symmetric (DES) and asymmetric (PKA) keys in a way that only the coprocessor can use them. There are two methods of storing keys for use with the coprocessor. The first method is storing the keys in key store files outside the coprocessor. These keys are encrypted under the master key of the coprocessor. When the coprocessor has to perform tasks using the externally stored keys, it reads the encrypted key into the coprocessor adapter and decrypts it with the master key, which ensures that the DES and PKA keys are not available in a usable form outside the coprocessor. This method is required if you want to use the keys by multiple coprocessors, for example, for load sharing.

Certain companies, such as government agencies or financial institutions have security requirements that dictate that the keys must remain in the coprocessors tamper-responding module at all times and are never allowed to leave the coprocessor for whatever reason. These keys are called retained keys. This is the method to operate the coprocessor in the most secure manner.

Considering the previous information, you understand that backing up information that is used in cryptographic devices is different from backing up operating system and user files.

4.12.1 Keys stored in key store files

Key store files are physical files stored in a library in the QSYS.LIB file system. To restore the keys and be able to use them after a disaster recovery or in a new system, you need to do the following with your resources and data:

- Save all key store files.
- Keep the coprocessor's master key in a very secure place, preferably using a master key split in three parts. This option is only available when the master key was manually entered. If you have chosen to auto-generate the master key, you have no way of backing the master key up except for cloning it to another coprocessor.
- Use the 4758 Cryptographic Coprocessor configuration utility and perform the Display function and then print the displayed information.
- Keep a log of all the configuration changes, such as roles modified or profiles changed to be able to recreate the same coprocessor configuration on a new 4758 Cryptographic Coprocessor adapter. You may also choose to write your own applications using the appropriate APIs to perform configuration changes. In this case you would have the configuration information in the application or on a planning sheet.

To restore the keys and recreate the coprocessor configuration, you need to:

- Restore all key store files
- Use the 4758 Cryptographic Coprocessor basic configuration wizard to create the basic setup based on the printed information.
- If you used your own applications to perform coprocessor configuration tasks, run the applications now.
- Load and set the master key you have written down and kept in a secure place. If you used the auto-generate option for creating a master key, you need to clone the master key from the coprocessor that holds the backed-up master key.

The essential steps you need to do to use the keys that are stored in key store files are to just restore the key store files and set the same master key on the coprocessor that was used to encrypt the keys.

4.12.2 Keys stored in the tamper-responding module

Since this is the most secure manner of operating the 4758 Cryptographic Coprocessor and storing the DES and PKA keys, it does not allow you to back up keys. That means the only way to recover from a disaster is to, for example, obtain new certificates and specify to store the private key in the hardware.

Chapter 5. Securing OS/400 application traffic with SSL/TLS

This chapter provides information about the current Secure Sockets Layer (SSL) support for TCP/IP applications. OS/400 V5R1 introduced many enhancements that simplify configuration tasks. Several additional server and client applications have been enabled for SSL. The chapter also guides you through the SSL configuration of the Telnet and FTP server. Once you have seen how these two servers are enabled for SSL, you can basically perform the same steps to set up other TCP/IP applications for SSL. More information about the SSL configuration for other TCP/IP applications can be found in:

- *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659
- *Digital certificate management* found in the iSeries Information Center by clicking **Security->Digital certificate management**
- *Securing applications with SSL* found in the iSeries Information Center by clicking **Security->Securing applications with SSL**

5.1 SSL/TLS support in OS/400

Since SSL first became available with OS/400 V4R1, more functions have been added over several OS/400 releases. Also OS/400 V5R1 introduces new SSL support for various server and client applications. In the meantime, the available OS/400 TCP/IP server applications are almost all enabled to support SSL V3.0 and TLS V1.0. While still many of the server applications support server authentication only, the number of applications that also support client authentication has been increased. Table 7 shows the SSL/TLS capabilities of the OS/400 TCP/IP applications. Note that the applications are marked as server or client applications. For example, the OS/400 TCP/IP Telnet server supports SSL with server and client authentication, but the OS/400 TCP/IP Telnet client is not SSL enabled yet.

Table 7. SSL capabilities of OS/400 TCP/IP server applications

SSL enabled application	Server authentication	Client authentication
Client Access Express (Host Servers) 1	X	
Management Central Server	X	
OS/400 DRDA/DDM Server	X	
Host On-Demand	X	X

SSL enabled application	Server authentication	Client authentication
HTTP Server (Original) and (powered by Apache)	X	X
Java applications	X	
LDAP Directory Services Server	X	X (New)
LDAP Directory Services Client	X (New)	X (New)
LDAP Directory Services Publishing	X	X (New)
Webserver Search Engine	X (New)	
OS/400 TCP/IP Telnet Server 2 3	X	X (Changed)
OS/400 TCP/IP FTP Server 3	X (New)	X (New)
OS/400 Cluster Security	X (New)	X (New)

1 Client Access Express allows SSL protection based on single remote connections. User-written applications that use Client Access Express APIs can also be protected by SSL.

2 Currently the IBM Personal Communications V.5.0 product, the 5250 emulation of the Client Access Express product, and Host On-Demand support the configuration for SSL connections. Client authentication is supported in V5R1 5250 Emulation, PCOMM V.5.0, and Host On-Demand 4.0 and higher.

3 SSL is supported only for the server application. The OS/400 TCP/IP Telnet and FTP client do not support SSL yet. Client authentication for these server applications requires that the certificates presented by the client are associated with an OS/400 user profile.

5.2 Enabling SSL for Telnet

The OS/400 TCP/IP Telnet server supports both server and client authentication. There are three major Telnet client products that support SSL with client authentication:

- PC5250 included in the Client Access Express
- IBM Personal Communications V5.0
- IBM WebSphere Host On-Demand

  You can find more information on how to enable the Host On-Demand client for client authentication in *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149.

Enabling client authentication for the Telnet server requires all clients to present a certificate during SSL session establishment. The certificate must be associated with an OS/400 user profile. In this case, clients that do not support client authentication cannot connect to the secure Telnet server. However, if configured, unsecure connections can still be established through the Telnet server port 23.

The following table shows the port numbers used for Telnet connections.

Table 8. The Telnet server port numbers

Connection	Port number	Service name
No SSL	23	telnet
SSL	992	telnet-ssl

Implementation tasks overview

The implementation tasks to enable SSL for IBM Personal Communications V5.0 are as follows:

1. Be sure that you have all of the required license programs installed.

   Refer to 2.1.1, “Installation prerequisites” on page 12, for the list of required license programs.

2. Prepare a CA certificate and server certificate.

   Refer to Chapter 2, “Digital Certificate Manager” on page 11, for instructions on how to create certificates on the iSeries or AS/400 system. Refer to Appendix D, “Creating a local Certificate Authority” on page 429, for instructions on how to set up a local Certificate Authority.

3. Set up the OS/400 TCP/IP Telnet server to use SSL.

4. Set up IBM Personal Communications for SSL connections with client authentication.

5.2.1 Objective

The objective of this section is to show you how to configure the OS/400 TCP/IP Telnet server to use SSL with client authentication. This scenario uses the IBM Personal Communications V.5.0 product as the Telnet client software. A server certificate issued by a private CA is used for establishing SSL connection.

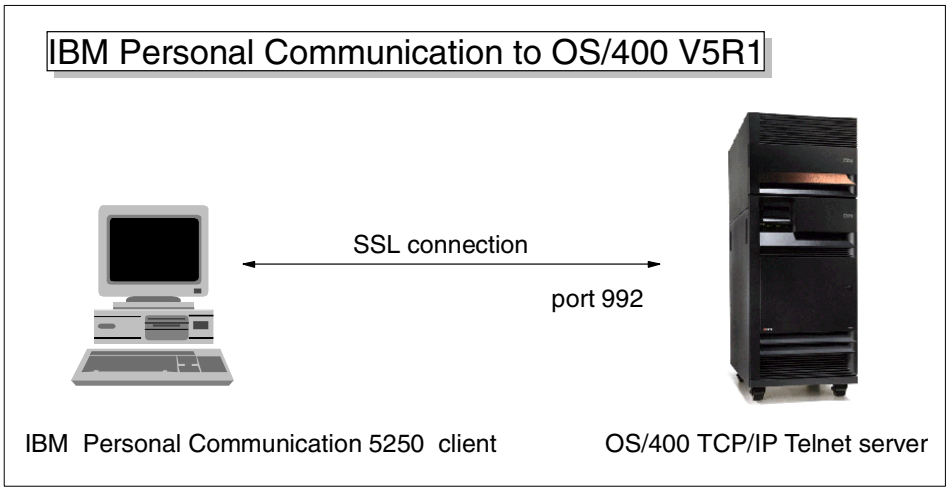


Figure 208. OS/400 TCP/IP Telnet server enabled for SSL

5.2.2 Configuring the iSeries server

You must enable SSL by setting the Telnet server attributes and associating a server certificate with the OS/400 TCP/IP Telnet server application ID. In the second step you need to update the application definition to enable client authentication. Perform the steps in this section to configure the OS/400 TCP/IP Telnet server for SSL with client authentication.

5.2.2.1 Enabling SSL for the Telnet server

Enabling SSL for the Telnet server as described in the following steps is necessary to open port 992 for SSL connections. However, without a server certificate being assigned to the Telnet server, a secure connection cannot be established. With V5R1 you can control whether the Telnet server supports only unsecure connections, only secure connections, or both. Perform the following steps to activate SSL support for the Telnet server:

1. Start the Operations Navigator.
2. Under the system name expand to **Network->Servers** and click **TCP/IP** to display the list of TCP/IP server applications.
3. Right-click the **TELNET** server icon to display the context menu as shown in Figure 209.

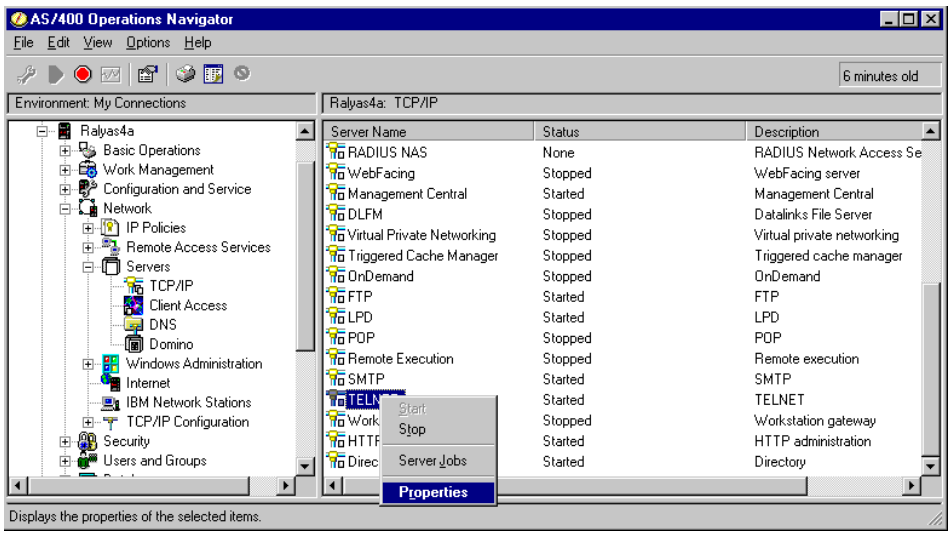


Figure 209. Operations Navigator expanded to Network->Servers->TCP/IP

4. Select **Properties**.
5. In the Properties window, click the **General** tab.

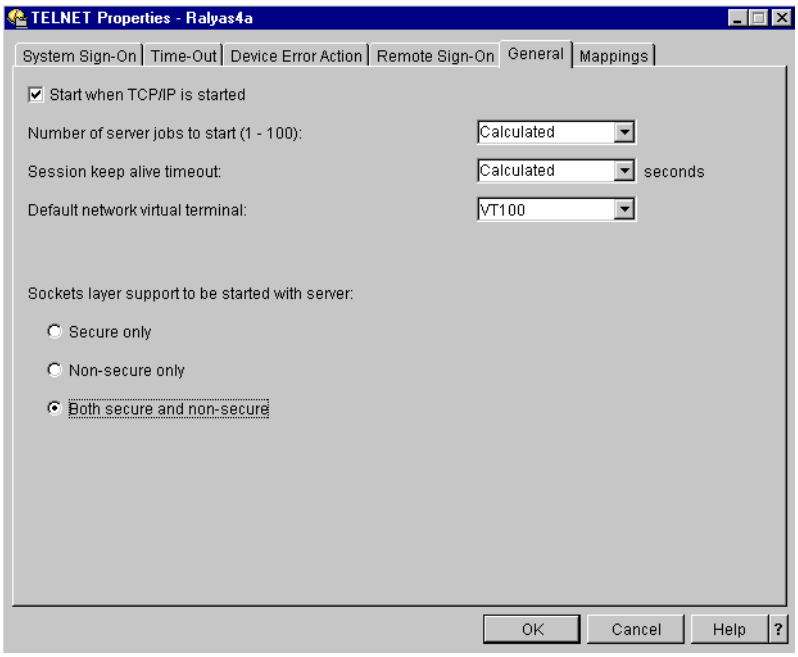


Figure 210. Telnet server properties: General tab

The new attributes allow you to control whether you allow unsecure, secure, or both types of connections to the Telnet server. You can set the attribute to the following values:

Secure only: When **Secure only** is selected, the server listens on port 992 only. No unsecure connections are accepted by the Telnet server.

Non-secure only: SSL connections are not allowed. Only unsecure connections are permitted through port 23.

Both secure and non-secure: Both SSL and non-SSL connections are allowed. Both the ports 23 and 992 are in Listen state.

6. Select **Both secure and non-secure** to enable the Telnet server for secure connections and at the same time allow non-secure connections. If your security policies require secure connections and your infrastructure is set up that all clients are also SSL-enabled you may want to operate the Telnet server for SSL connections only. Another alternative is to use OS/400 IP packet filtering to allow unsecure connections only from certain IP addresses or subnets.
7. Click **OK** to save the configuration changes.

5.2.2.2 Assigning a server certificate to the Telnet server

The following steps guide you through assigning an existing server certificate to the Telnet server. Refer to Chapter 2, “Digital Certificate Manager” on page 11, for more information about assigning certificates to applications and how to create certificates.

1. Start the Digital Certificate Manager through the Operations Navigator or the AS/400 Tasks page. You need a user profile with *ALLOBJ and *SECADM special authorities to perform the configuration.
2. Click **Select a Certificate Store,** then select the ***SYSTEM** certificate store and click **Continue**.
3. Enter the certificate store password and click **Continue**.
4. On the DCM navigation pane, click **Manage Applications**.
5. From the available options under the Manage Applications group, click **Update certificate assignment**.
6. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed as shown in Figure 211.

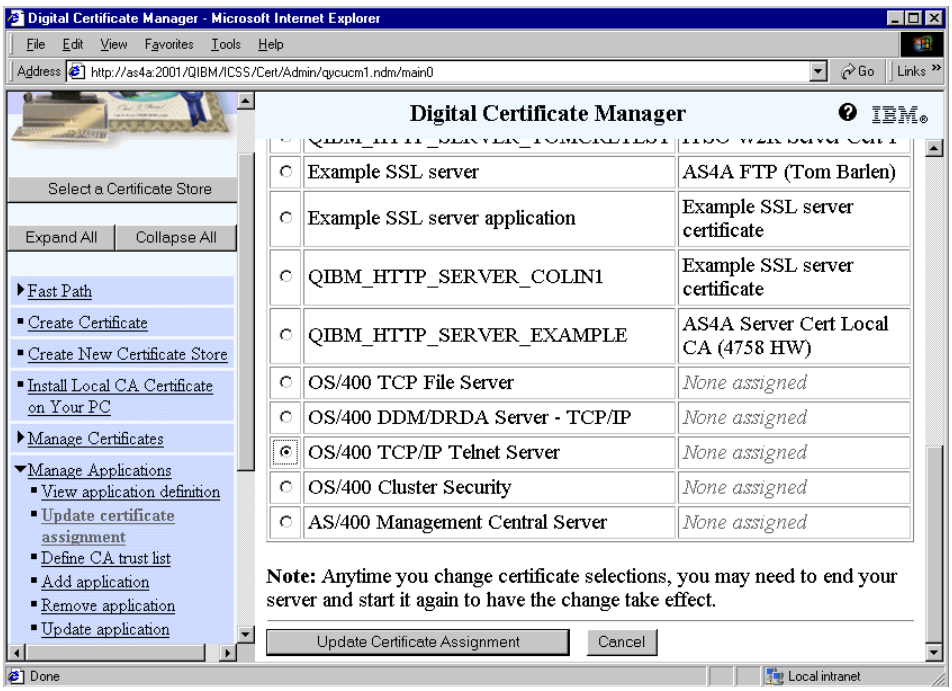


Figure 211. Digital Certificate Manager: Update Certificate Assignment window

7. Select the **OS/400 TCP/IP Telnet Server** and click **Update Certificate Assignment**. The list of available certificates is displayed.

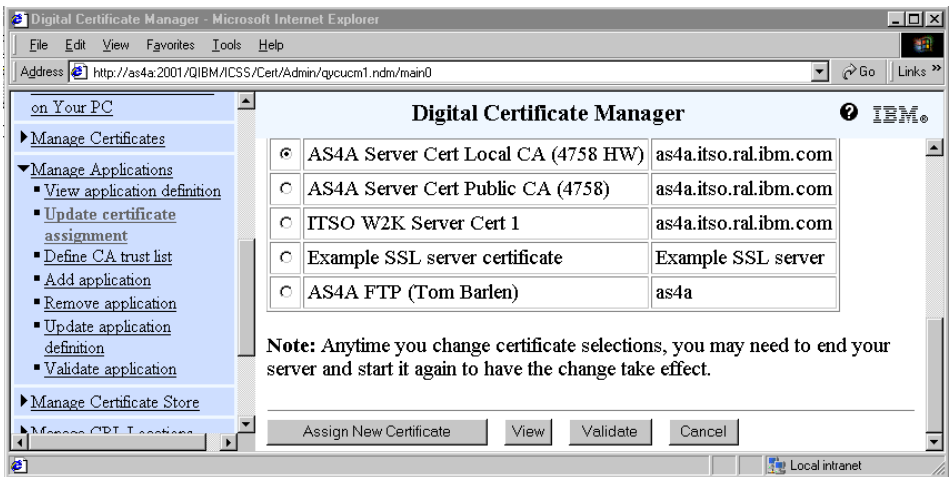


Figure 212. Digital Certificate Manager: Select a certificate

8. Select the certificate from the list that you want the Telnet server to use to establish SSL connections between the Telnet server and client.
9. Click **Assign New Certificate**.

   DCM displays a confirmation message as shown in Figure 213.

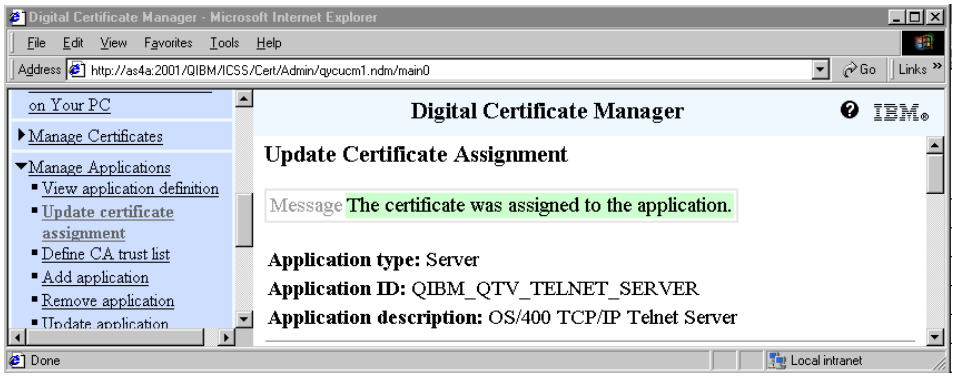


Figure 213. Digital Certificate Manager: Confirmation message

10. Click **Cancel** to return to the list of server applications.

5.2.2.3 Updating the application definition for client authentication

It is important to understand that when client authentication is set to Required in the Telnet server application definition, each client that wants to connect via a secure connection must present a certificate that is associated with an OS/400 user profile.

The Telnet server client authentication support was already made available to V4R4 by applying additional PTFs. A service program (QTVSSL) had to be called to enable or disable required client authentication. In V5R1 you can enable this support conveniently through the DCM configuration interface.

Perform the following steps to enable client authentication for the Telnet server:

1. Start DCM.
2. From the DCM navigation pane, click **Manage Applications**.
3. From the Manage Applications group, click **Update application definition**.
4. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed as shown in Figure 214.

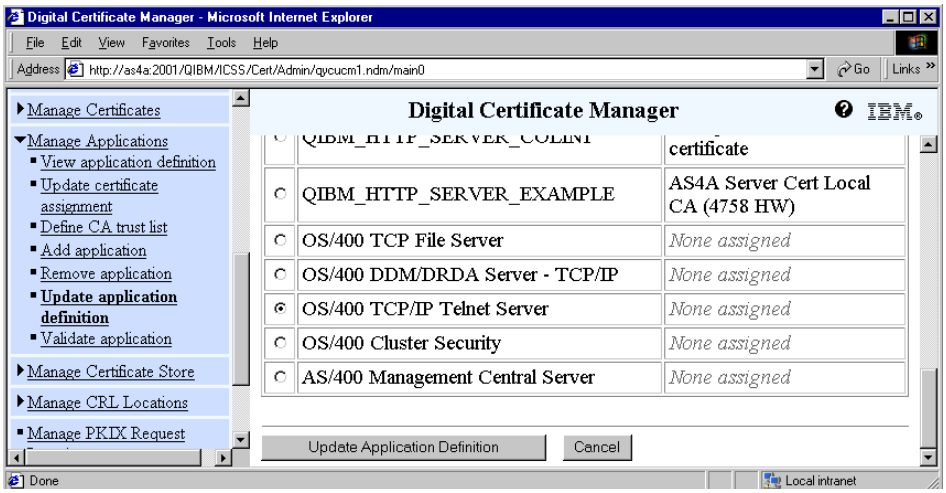


Figure 214. Digital Certificate Manager: Update Application Definition window

5. Select the **OS/400 TCP/IP Telnet Server** and click **Update Application Definition**.

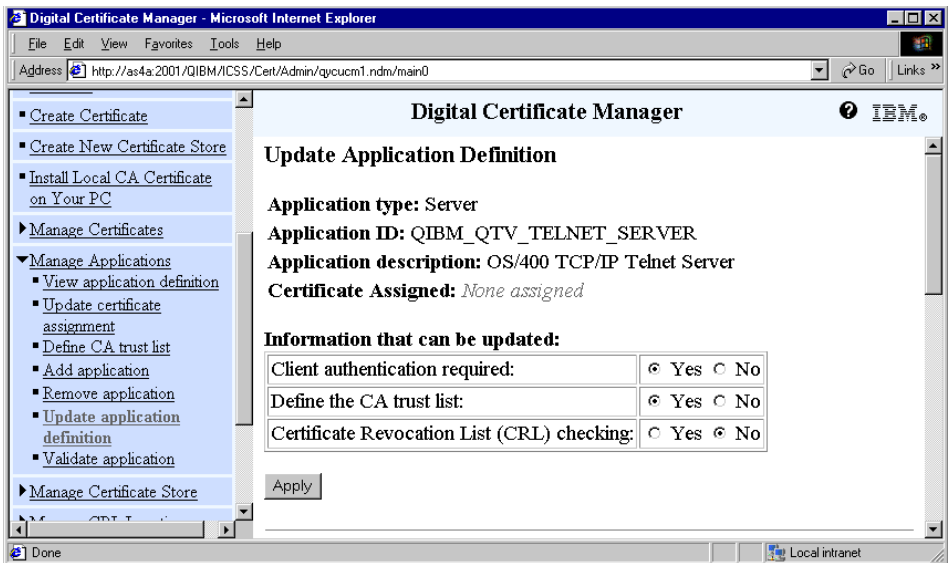


Figure 215. Digital Certificate Manager: Selecting client authentication

Mostly, when referring to enabling the Telnet server for client authentication, it means to configure the server to require client authentication. The Telnet server application itself is always enabled for

client authentication; it just does not require it. As mentioned before, when you configure the server so that clients must authenticate themselves by presenting a certificate, clients without a valid certificate cannot establish a secure connection. Be careful when configuring the Telnet server for required client authentication.

6. Set **Client authentication required** to **Yes** and click **Apply**.
7. A DCM confirmation message is displayed. Click **Cancel** to return to the list of server applications.

5.2.2.4 Update the CA trust list

In this last task of the server setup, you need to specify which client certificates the Telnet server accepts for client authentication. That is, the client certificates must be issued by a certificate authority that the Telnet server trusts.

1. From the Manage Applications group, click **Define CA trust list**.
2. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed.
3. Select the **OS/400 TCP/IP Telnet Server** and click **Define CA Trust List**.

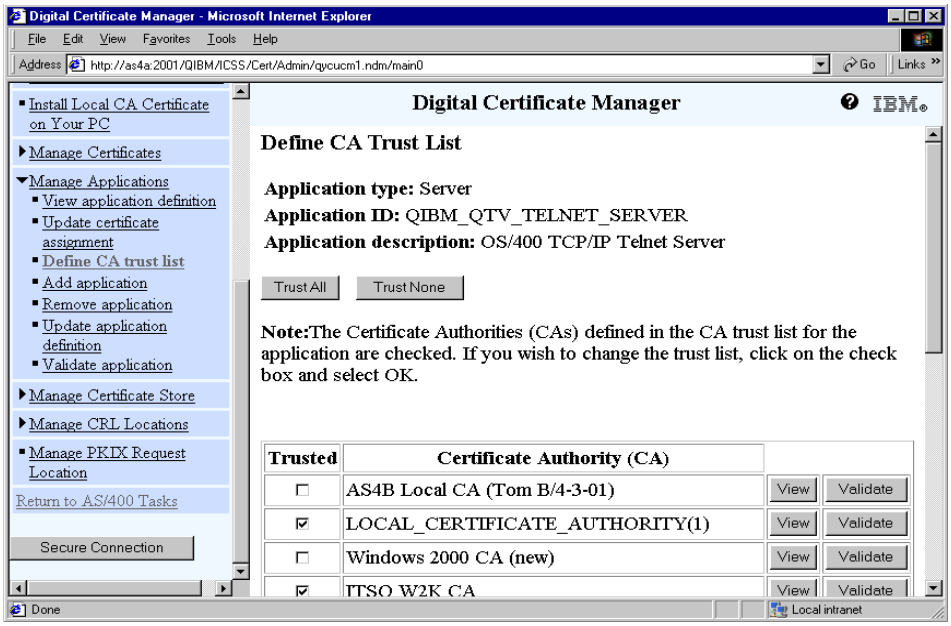


Figure 216. Digital Certificate Manager: Define CA Trust List

In the Define CA Trust List window, define the client or user certificates that this server application accepts for client authentication. In other words, only client or user certificates that are issued by a trusted CA are accepted by the server application. The list of displayed CAs contains only those CAs that are *Enabled* in the storewide CA list as defined in the Work with CA certificates option from the Fast Path menu. This means, if you installed a CA certificate, and it is not shown on the Define CA Trust List window, you must ensure that the CA is enabled.

4. Select all CAs that issued user or client certificates you want the Telnet server to accept. Note that these certificates still need to be associated with an OS/400 user profile.
5. Click **OK**. The list of CAs is displayed again with a confirmation message stating that the CAs have been trusted.
6. Click **Done** to return to the list of server applications.
7. Restart the Telnet server.

   To activate these changes to the Telnet server, you must restart it using the following commands:

   - To stop: `ENDTCPSVR SERVER(*TELNET)`
   - To start: `STRTCPSVR SERVER(*TELNET)`

Note

Take extra care when ending the Telnet server on the AS/400 system. All active Telnet sessions will also be ended.

5.2.3 Configuring the PC

First, you must obtain a user or client certificate for the IBM Personal Communications client. Every time the client establishes a connection the certificate must be presented to the OS/400 TCP/IP Telnet server.

Next, you must import the private CA certificate into the key database file. Note that if you use server and user certificates issued by a well-known CA, you do not need to exchange CA certificates. They are usually preloaded on all standard software applications.

The IBM Personal Communications (PComm) product has its own key database file. After the CA certificate is imported, you have to import the

client or user certificate and then configure the particular emulation session to use SSL.

5.2.3.1 Obtaining a user certificate and CA certificate

In this scenario we use a local OS/400 CA to create a user certificate. The certificate will then automatically be associated with the user profile that signed on to DCM. You could also obtain a certificate from a well-known CA and manually associate the certificate using DCM to a particular OS/400 user profile. The following steps show you how to create a user certificate:

1. Start DCM.
2. When prompted sign on with a user profile you want to create the user certificate for. In this scenario we used a user profile without any special authorities.
3. From the DCM navigation pane, click **Create certificate**.

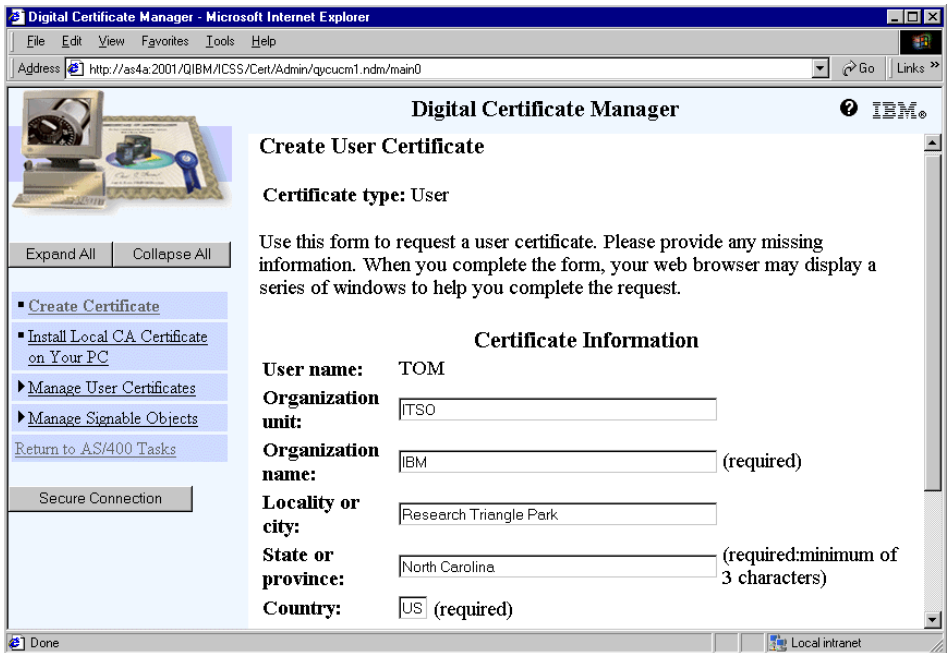


Figure 217. Digital Certificate Manager: Create User Certificate

4. Enter the data for the certificate's distinguished name. Note that the user name is already filled in and cannot be altered. This assures that the new certificate will automatically be associated with this user profile.

5. Click **Continue**. After the certificate is created, the following window is displayed. Note that the public/private key pair is generated in the PC's Web browser.

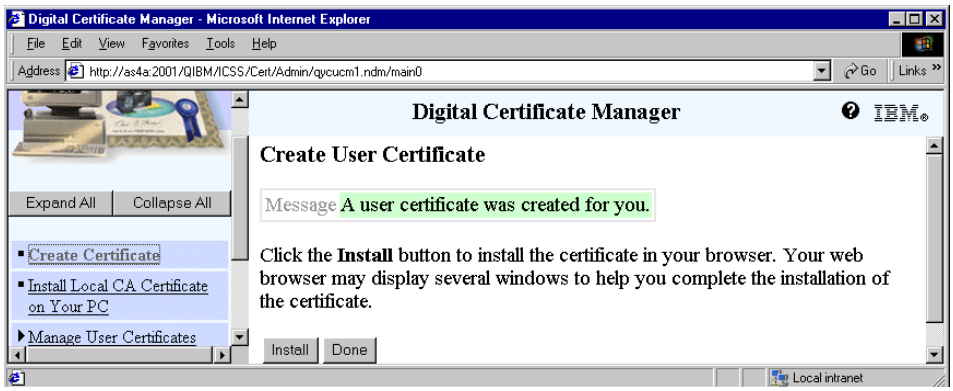


Figure 218. Digital Certificate Manager: Create User Certificate completion message

6. Click **Install** to install the signed user certificate into the browser's key database. A confirmation message is displayed. Click **OK** to continue. Note the messages vary depending on the browser version used to create the certificate.
7. Click **Done** to complete the certificate creation process.

   The user certificate is created and installed in the browser's key database. In the next few steps you will download the local CA certificate onto your PC.
8. From the DCM navigation pane, click **Install Local CA Certificate on Your PC**.

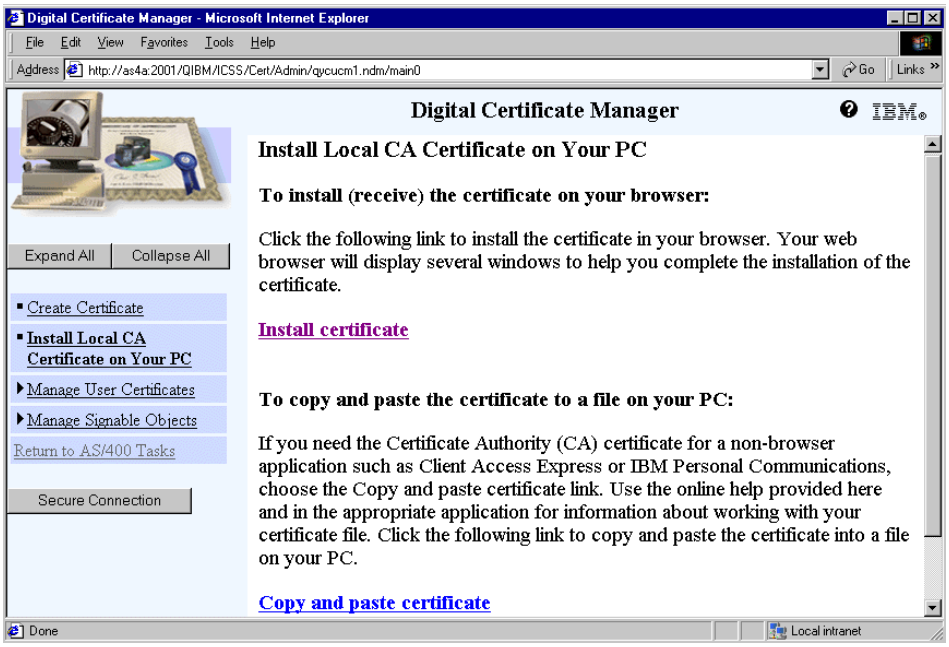


Figure 219. Digital Certificate Manager: Install Local CA Certificate on Your PC

9. Click **Install certificate**. Microsoft's Internet Explorer does not install the CA certificate into its key database. Instead it downloads the file containing the CA certificate as shown in Figure 220.

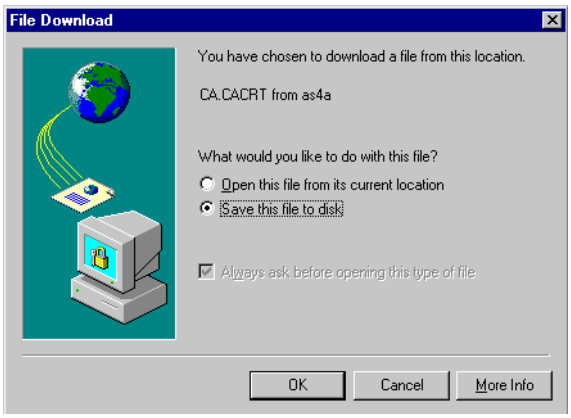


Figure 220. File Download window

10. Select **Save this file to disk** and click **OK**.

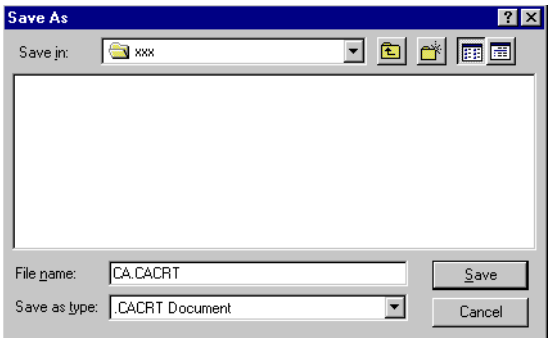


Figure 221. Save as window

11. Specify a location where the CA certificate should be stored and click **Save**.

To use the new user certificate with IBM Personal Communications, you need to export it from the browser's key database. In this scenario, we used the Microsoft Internet Explorer 5.01.

12. From the browser window's action bar, click **Tools->Internet Options**.
13. Click the **Content** tab.
14. Click **Certificates**.

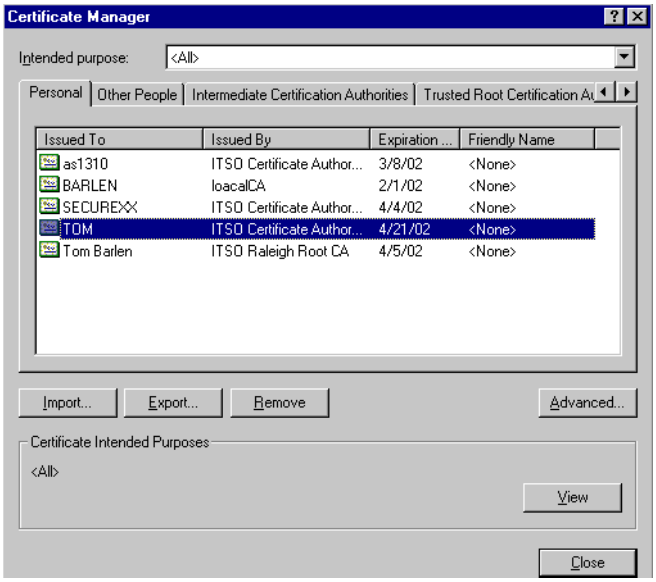


Figure 222. Internet Explorer: Certificate Manager

15. Select the user certificate that you just created and click **Export**. A Certificate Manager Export Wizard starts.
16. On the first window of the wizard, click **Next**.

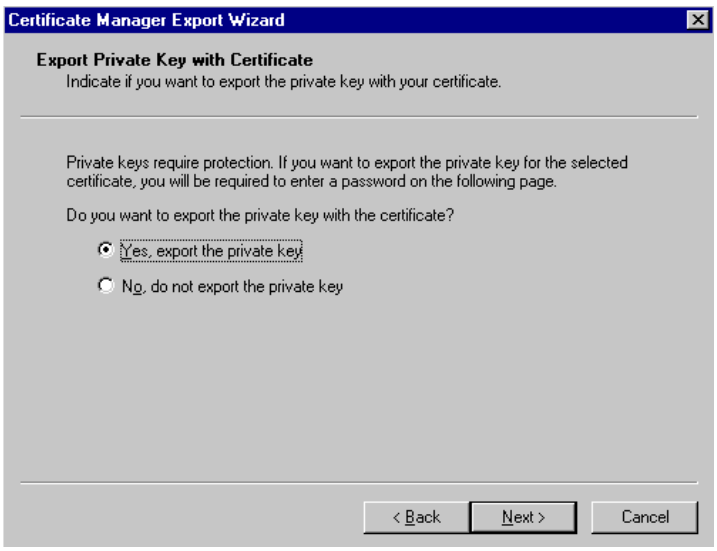


Figure 223. Internet Explorer: Certificate Manager Export Wizard

17. Select **Yes, export the private key** and click **Next**.

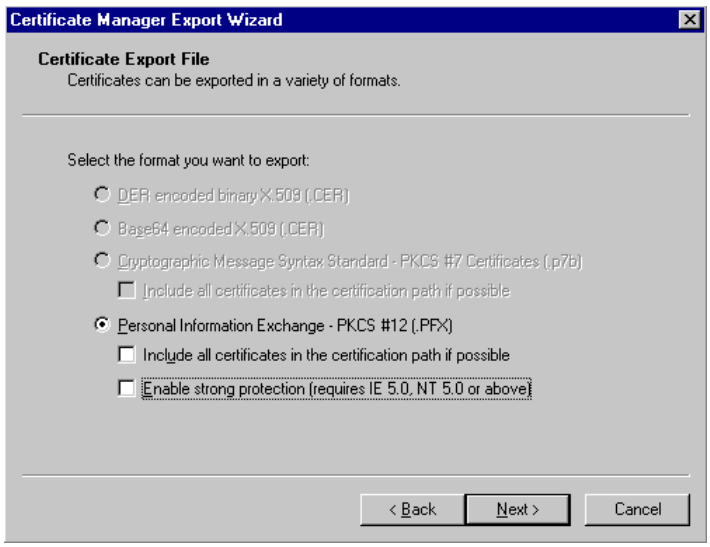


Figure 224. Internet Explorer: Certificate Manager Export Wizard

18. Clear the **Enable strong protection (requires IE 5.0, NT 5.0 or above)** checkbox and click **Next**.

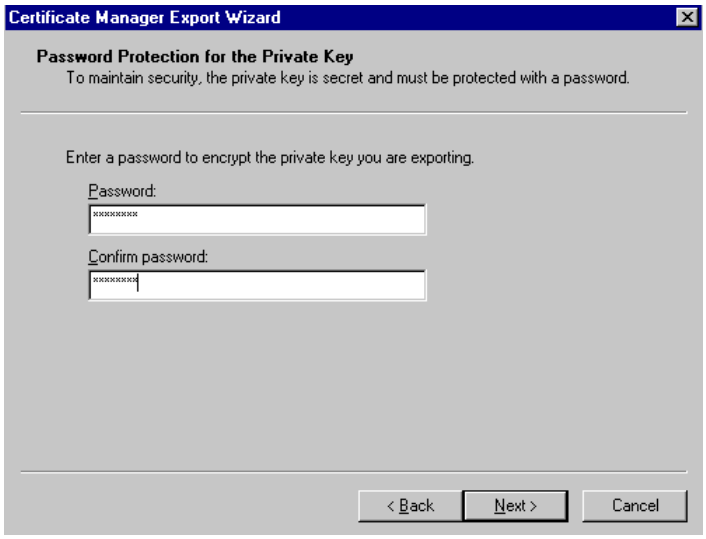


Figure 225. Internet Explorer: Certificate Manager

19. Enter a password. This password is used to encrypt the exported certificate including the private key. You need this password later when importing the certificate into the IBM Personal Communications key database.
20. Click **Next**.
21. Enter a path and file name to store the exported certificate and click **Next**.

    In this scenario, the following path and file name was used:

    `C:\xxx\tom.pfx`
22. Click **Finish** to compete the certificate export. A confirmation message is displayed. Click **OK** to close the message window.
23. Click **Close** to close the Certificate Manager and then **OK** to close the Internet Options window.

5.2.3.2 Importing a CA certificate to the key database

Perform the following steps to add a private CA certificate to the key database:

1. Start the IBM Key Management program in the IBM Personal Communications folder using the following default path:

Start->Programs->IBM Personal Communications->Utilities->Certificate Management

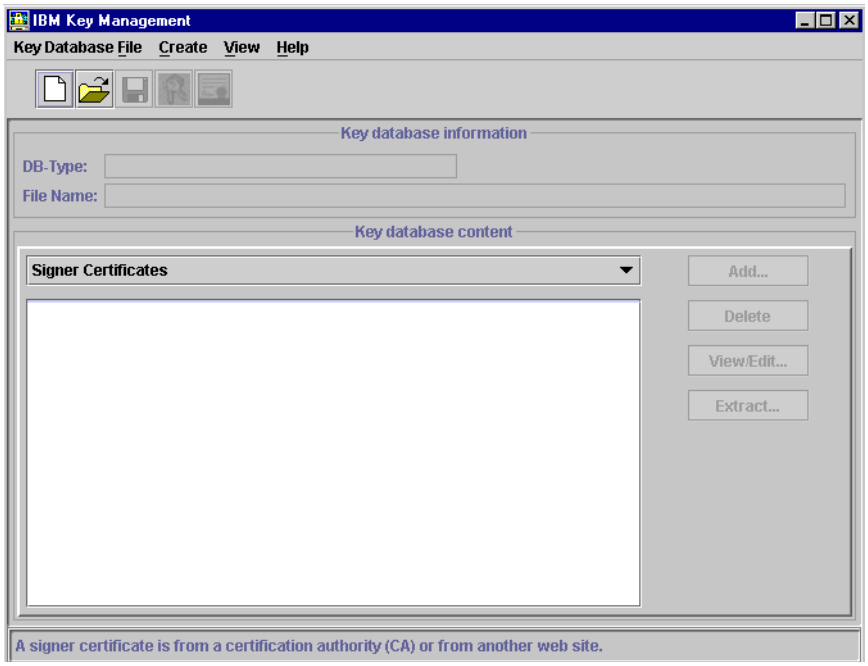


Figure 226. IBM Key Management

2. Click **Key Database File->Open**.

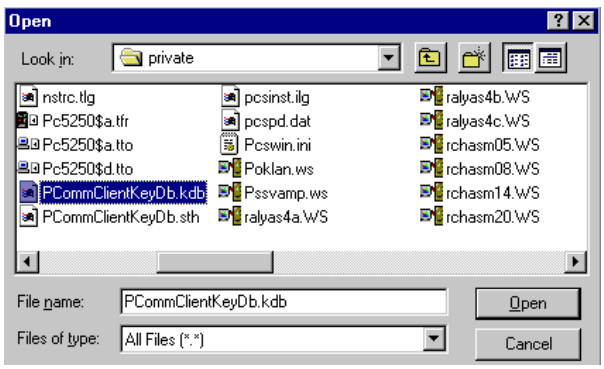


Figure 227. Open a default key database file

3. Select the key database file **PCommClientKeyDb.kdb** and click **Open**.

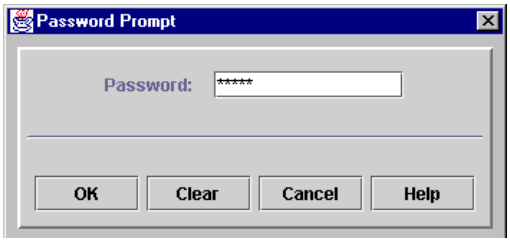


Figure 228. Password prompt

4. Enter the password and click **OK**. By default, the password is pcomm.

   The currently installed CA certificates are displayed.

5. Click **Add** on the right side of the Key Management window to add a new CA certificate to the key database.

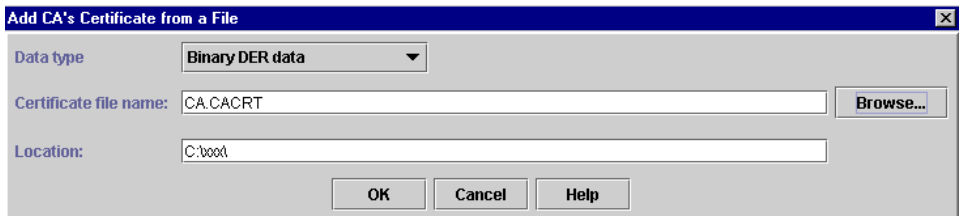


Figure 229. Import file selection window

6. Enter the path name and the file name of the CA certificate you want to import and click **OK**.

Figure 230. Certificate label name input window

7. Enter a label for the CA certificate and click **OK**.

   The certificate is stored as a trusted root in the key database file.

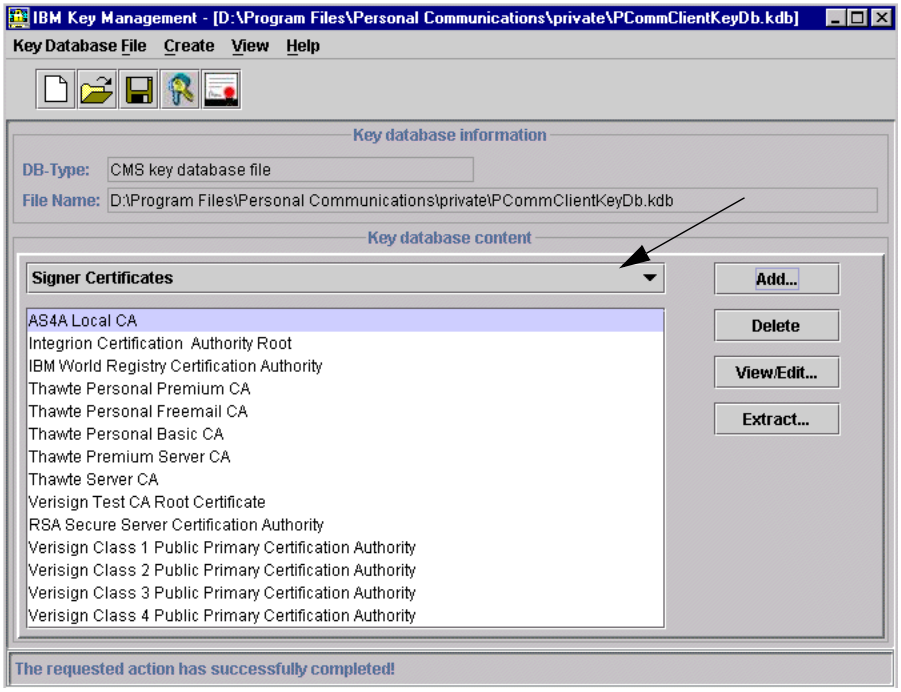


Figure 231. IBM Key Management main window

8. Click **View/Edit** to confirm the imported certificate's contents.

   The key database on your PC now has the private CA certificate and it is registered as a trusted root. In the next steps you import the user certificate into the key database of IBM Personal Communications.

9. From the IBM Key Management window select the bar titled **Signer Certificates** and click **Personal Certificates**.

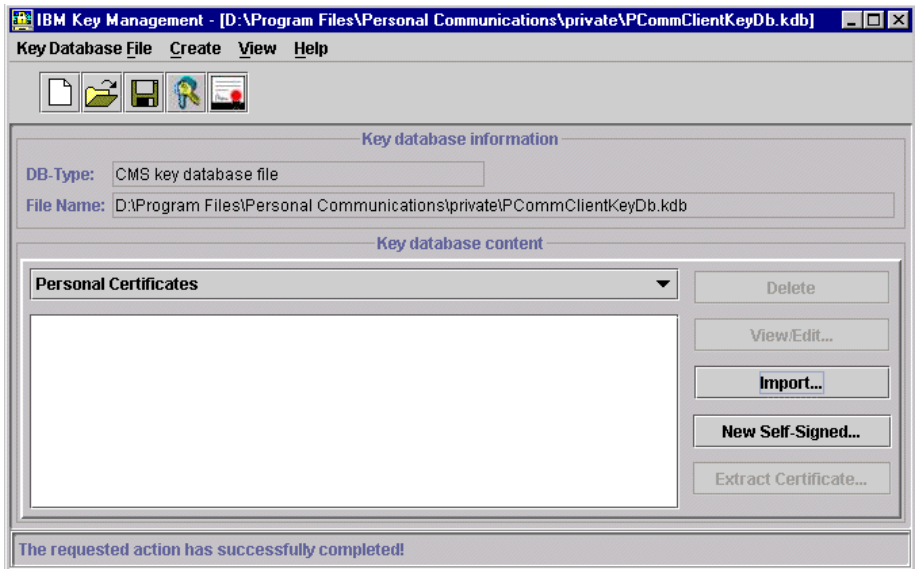


Figure 232. IBM Key Management: Personal Certificates

10. Click **Import**.

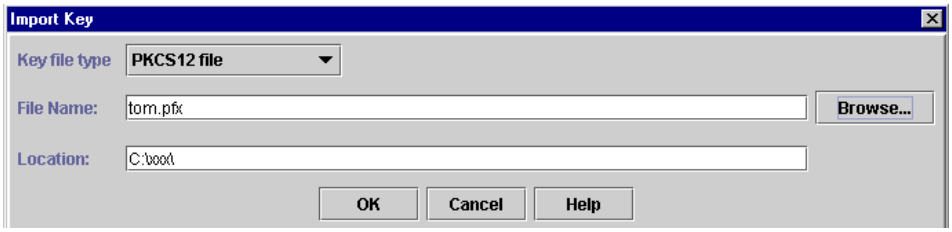


Figure 233. Import Key window

11. Enter the path and file name you specified when exporting the certificate from the Internet Explorer's key database and click **OK**.

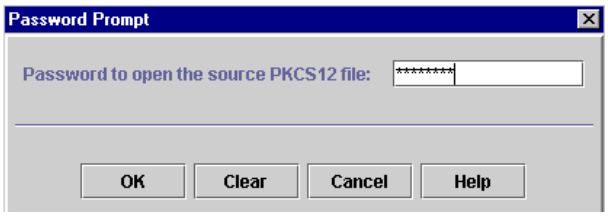


Figure 234. Password Prompt window

12. Enter the password that you specified when exporting the certificate and click **OK**.

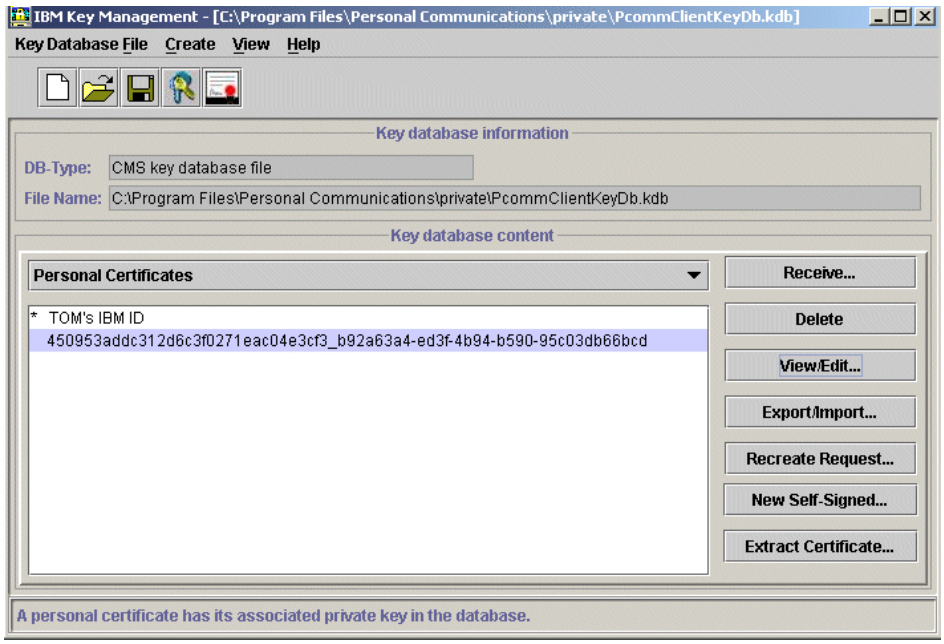


Figure 235. IBM Key Management window with imported personal certificate

In Figure 235, note the unusual format of the certificate label. This is a problem of Microsoft's Certificate Manager. Netscape's Navigator works fine and in the case of the certificate used in this scenario, the Netscape browser generated a label of TOM's IBM ID. We imported the TOM's IBM ID certificate first. This is the reason why it is shown in the list of imported personal certificates.

13. Click **Key Database File->Close**.

5.2.3.3 Configuring IBM Personal Communications for SSL

The following steps describe how to enable SSL with client authentication for an IBM Personal Communications 5250 session to an iSeries server:

1. Start a Personal Communications (PComm) session that you want to set up for SSL.
2. Click **Communication->Configure**.

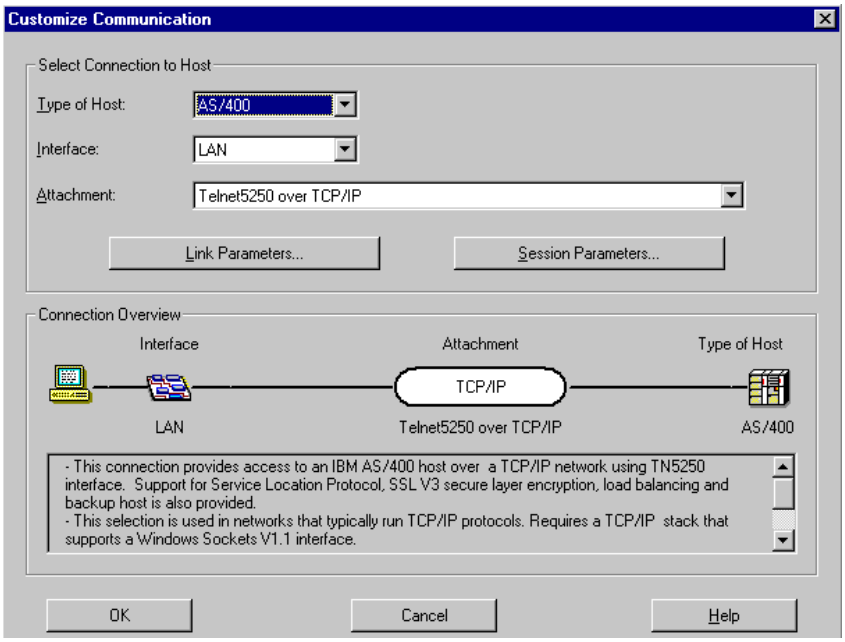


Figure 236. Customize Communication window.

3. Click **Link Parameters**.

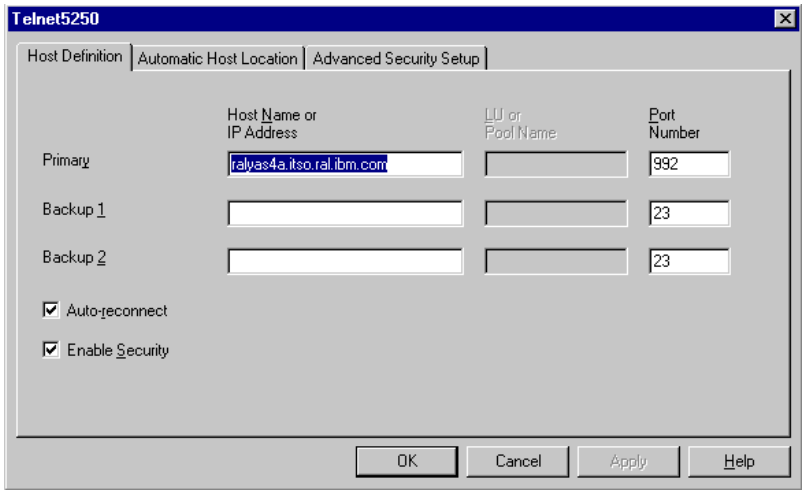


Figure 237. Telnet 5250 window

4. Change the port number from `23` to `992` and select the **Enable Security** check box. The emulation session is now configured for SSL with server

authentication only. In the next few steps you also enable client authentication.

5. Click the **Advanced Security Setup** tab. You will see Figure 238.

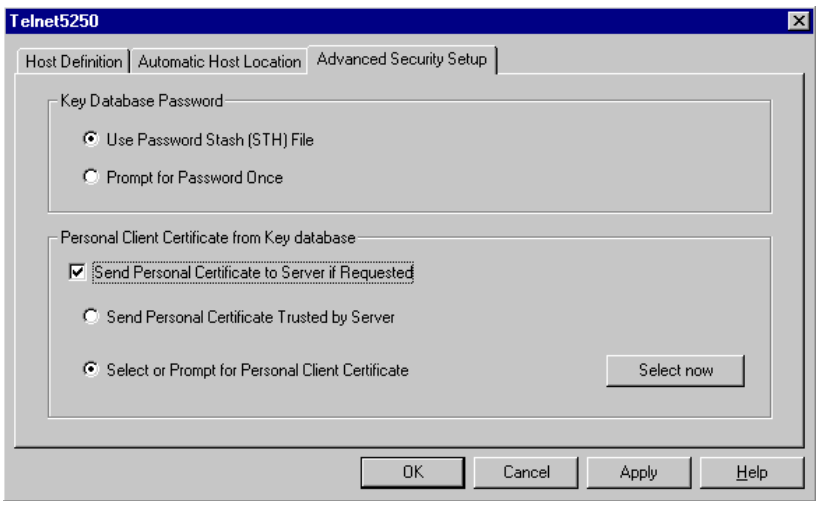


Figure 238. Telnet 5250 window: Advanced Security Setup tab

In the Key Database Password section you define whether the password to open the key database at connection time should be retrieved from the stashed password file or entered manually by the user. The password option defined in this section is only used to open the key database for verifying the CA trust. It is not used to retrieve client certificates.

6. Select **Use Password Stash (STH) File** and **Send Personal Certificate to Server if Requested**. Then select **Select or Prompt for Personal Client Certificate** and click **Select now**. By selecting this option, you can predefine which certificate you will present to the server when establishing a secure session.

   If you select **Send Personal Certificate Trusted by Server**, the client examines the received CA certificates that are sent by the server during an SSL handshake. It then automatically picks one that was issued by one of the received CA certificates and uses it for client authentication.

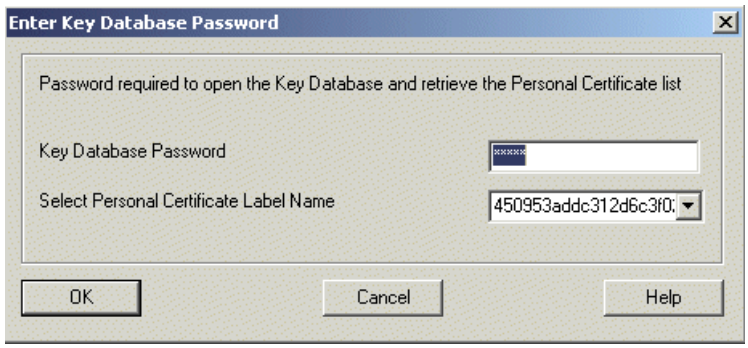


Figure 239. Enter Key Database Password window

7. Enter the key database password and select the client certificate you want to use for client authentication. Now you realize how important an easy-to-remember certificate label is.
8. Click **OK** three times to save the changes. In case you had an active session before, you also have to confirm the message indicating that the communication will be ended.

The session restarts automatically and establishes a secured connection. Remember to save the session profile with the new settings.

5.2.4 Verifying the SSL connection

The following procedures can be used to determine if the Telnet connection is protected using the SSL protocol.

On the iSeries server:

1. Use the `NETSTAT *CNN` command and check that the TCP port 992 is in the Listen state:

```
Remote          Remote  Local
Address          Port    Port  Idle Time  State
*                    *    992  000:21:47 Listen
```

2. When a Telnet session is established, use the `NETSTAT *CNN` command to check that the Telnet client has a connection through port 992 and not 23:

```
Remote          Remote  Local
Address          Port    Port  Idle Time  State
9.24.105.67           1308    992  000:00:00  Established
```

On the PC:

1. Start a Telnet session that is configured for SSL. Check the status bar at the bottom of the 5250 session window:.

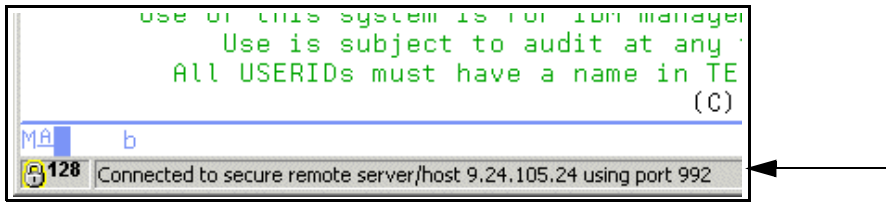


Figure 240. 5250 session using a SSL connection

The padlock indicates that the 5250 emulation is secured using the SSL protocol. In addition, the message shows that port 992 is used for this connection. You can also see the encryption key length used.

2. From the action bar, click **Communication->Security->Server**.

   You get information about the certificate the server presented to the client and the encryption algorithm and key length used to protect the session traffic, as shown in Figure 241.

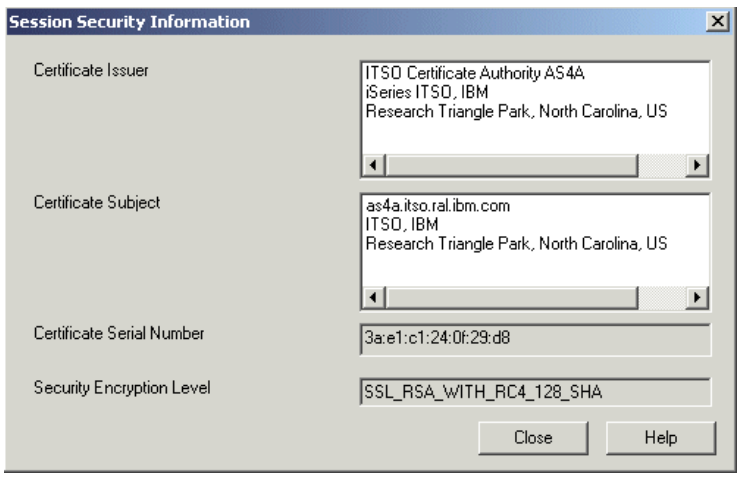


Figure 241. Personal Communications 5.0: Server security information

In this session, a cipher suite of RC4 encryption with 128-bit key length and SHA authentication algorithm is used.

3. Click **Close** and then click **Communication->Security->Client**. Now, the client certificate information is shown.

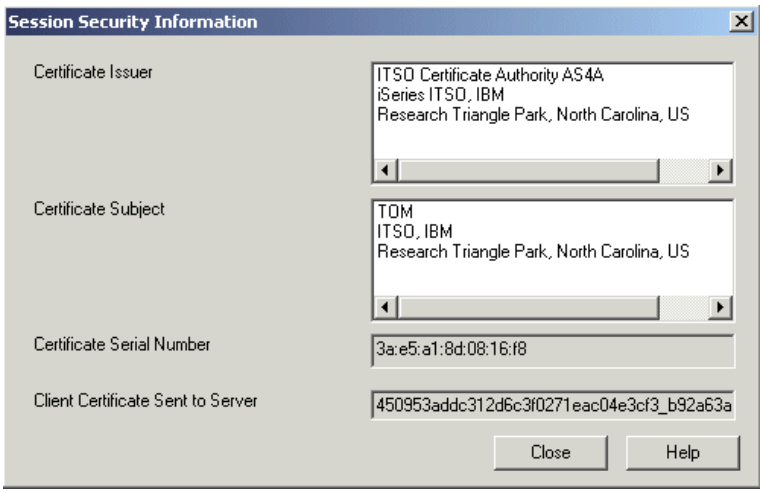


Figure 242. Personal Communications 5.0: Client security information

4. Click **Close**.

5.3 Enabling SSL for FTP

In V5R1 also the OS/400 TCP/IP FTP server is enabled for SSL and supports both server and client authentication. However, the OS/400 TCP/IP FTP client does not support SSL yet. We tested the following two PC-based FTP clients with success:

- BlueZone Secure FTP client from Seagull Software
- WS_FTP Pro 6.7 from Ipswitch

Enabling client authentication for the FTP server requires all clients to present a certificate during SSL session establishment. The certificate must be associated with an OS/400 user profile. In this case, clients that do not support client authentication cannot connect to the secure FTP server. However, if configured, unsecure connections can still be established through the FTP server port 21.

The following table shows the port numbers used for FTP connections.

Table 9. The FTP server port numbers

Connection	Port number	Service name
No SSL	21	ftp-control
SSL	21 and 990	ftps-control

There are two methods to establish a secure FTP connection. One uses port 990 and starts immediately with an SSL handshake. The BlueZone Secure FTP client uses this method. The second method uses port 21 and initiates an SSL handshake by issuing an `AUTH FTP` subcommand. The WS_FTP Pro client uses the latter method.

In this scenario we document the BlueZone FTP Client setup.

Implementation tasks overview

The implementation tasks to enable SSL for the OS/400 TCP/IP FTP server and the BlueZone Secure FTP client are as follows:

1. Be sure that you have all of the required license programs installed.

   Refer to 2.1.1, “Installation prerequisites” on page 12, for the list of required license programs.

2. Prepare a CA certificate and server certificate.

   Refer to Chapter 2, “Digital Certificate Manager” on page 11, for instructions on how to create certificates on the iSeries or AS/400 system. Refer to Appendix D, “Creating a local Certificate Authority” on page 429, for instructions on how to set up a local Certificate Authority.

3. Set up the OS/400 TCP/IP FTP server to use SSL.

4. Set up BlueZone Secure FTP client for SSL connections with client authentication.

5.3.1 Objective

The objective of this section is to show you how to configure the OS/400 TCP/IP FTP server to use SSL with client authentication. This scenario uses the BlueZone Secure FTP client product as the FTP client software. A server certificate issued by a private CA is used for establishing an SSL connection.

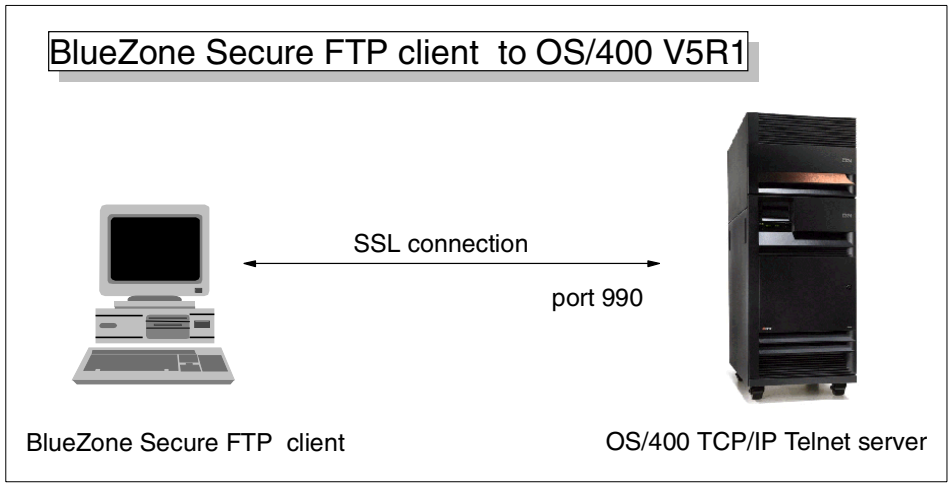


Figure 243. OS/400 TCP/IP FTP server enabled for SSL

5.3.2 Configuring the iSeries server

You must enable SSL by setting the FTP server attributes and associate a server certificate with the OS/400 TCP/IP FTP server application ID. In the second step you need to update the application definition to enable client authentication. Perform the steps in this section to configure the OS/400 TCP/IP FTP server for SSL with client authentication:

5.3.2.1 Enabling SSL for the FTP server

Enabling SSL for the FTP server as described in the following steps, is necessary to open port 990 and port 21 for SSL connections. However, without a server certificate being assigned to the FTP server, a secure connection cannot be established. With V5R1 you can control whether the FTP server supports only unsecure connections, only secure connections, or both. Perform the following steps to activate SSL support for the FTP server:

1. Start the Operations Navigator.
2. Under the system name, expand to **Network->Servers** and click **TCP/IP** to display the list of TCP/IP server applications.
3. Right-click the **FTP** server icon to display the context menu as shown in Figure 244.

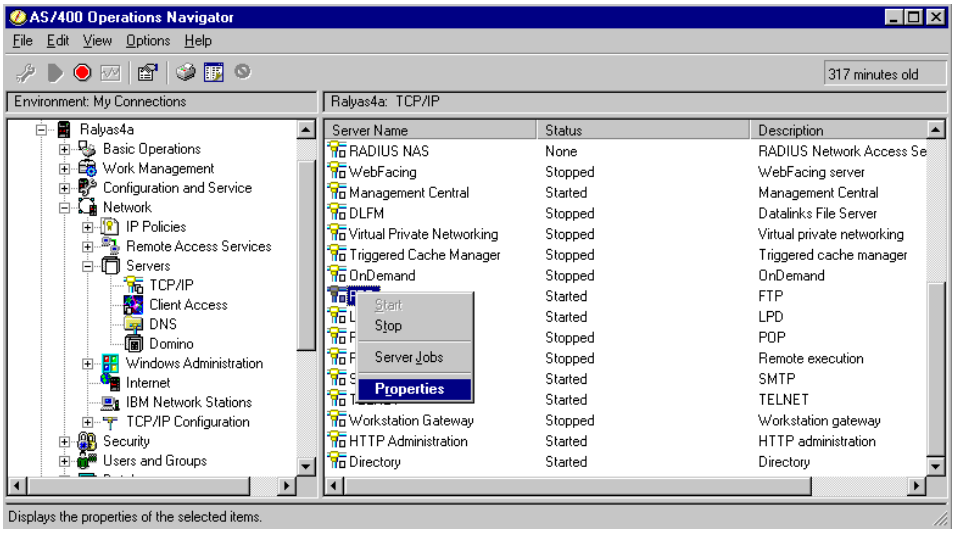


Figure 244. Operations Navigator expanded to Network->Servers->TCP/IP

4. Select **Properties**.

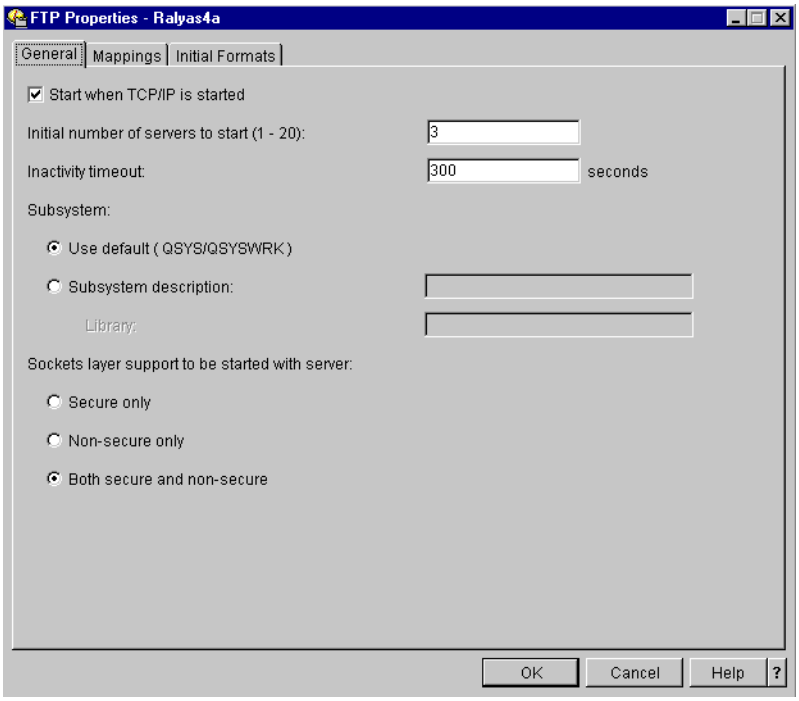


Figure 245. FTP server properties: General tab

The new attributes allow you to control whether you allow unsecure, secure, or both types of connections to the FTP server. You can set the attribute to the following values:

Secure only: When **Secure only** is selected, the server listens on port 990 and still listens on 21! However, if a connection is made to port 21, the user is not allowed to log in unless the client negotiates a secure control connection using the AUTH FTP subcommand (unless it's an anonymous FTP logon, in which case non-secure is allowed). Four your information, a connection to port 990 has an implied `AUTH SSL` subcommand in the "state diagram" for the server.

Non-secure only: SSL connections are not allowed. That is, the `AUTH FTP` subcommand is rejected.

Both secure and non-secure: Both SSL and non-SSL connections are allowed. An FTP client can connect directly to a "secure FTP" port (TCP port 990), in which case the session is set up with SSL/TLS initially. Another connection method is for the FTP client to connect to the FTP server on the regular (non-encrypted) port (TCP port 21) and then negotiate authentication and encryption options (by sending commands to the FTP server and interpreting the server's responses to those commands). In fact, FTP listens on both ports (21 and 990) but it only allows a connection through port 990.

5. From the properties General tab, select **Both secure and non-secure** to enable the FTP server for secure connections and at the same time allow non-secure connections. If your security policies require secure connections and your infrastructure is set up that all clients are also SSL-enabled, you may want to operate the FTP server for SSL connections only. Another alternative is to use OS/400 IP packet filtering to allow non-secure connections only from certain IP addresses or subnets.
6. Click **OK** to save the configuration changes.

5.3.2.2 Assigning a server certificate to the FTP server

The following steps guide you through assigning an existing server certificate to the FTP server. Refer to Chapter 2, “Digital Certificate Manager” on page 11, for more information about assigning certificates to applications and how to create certificates.

1. Start the Digital Certificate Manager through the Operations Navigator or the AS/400 Tasks page. You need a user profile with *ALLOBJ and *SECADM special authorities to perform the configuration.
2. Click **Select a Certificate Store,** then select the ***SYSTEM** certificate store and click **Continue**.
3. Enter the certificate store password and click **Continue**.
4. On the DCM navigation pane, click **Manage Applications**.
5. From the available options under the Manage Applications group, click **Update certificate assignment**.
6. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed as shown in Figure 246.

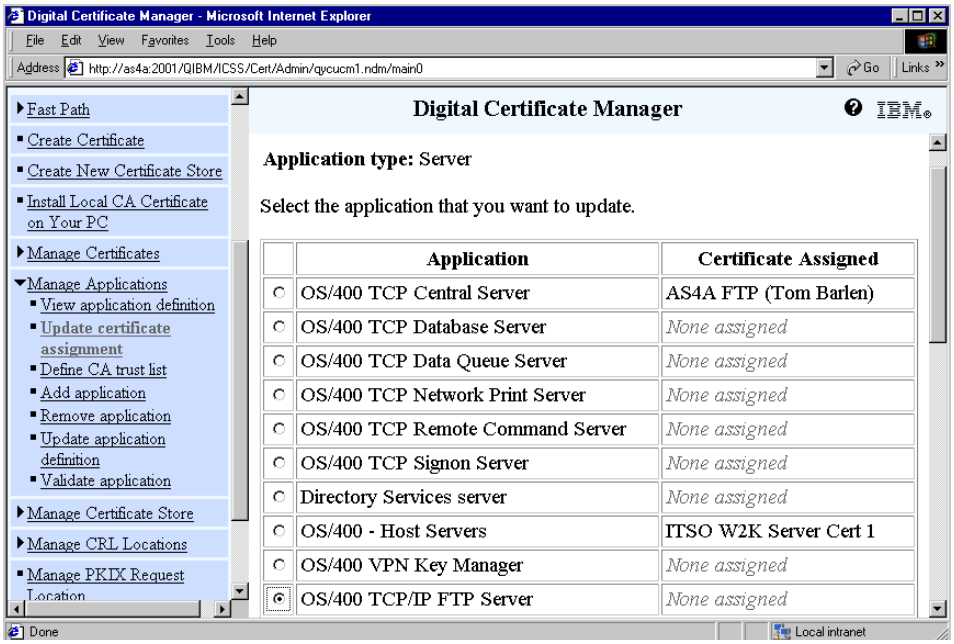


Figure 246. Digital Certificate Manager: Update Certificate Assignment window

7. Select the **OS/400 TCP/IP FTP Server** and click **Update Certificate Assignment**. You may have to scroll down to find the Update Certificate Assignment button. The list of available certificates is displayed.

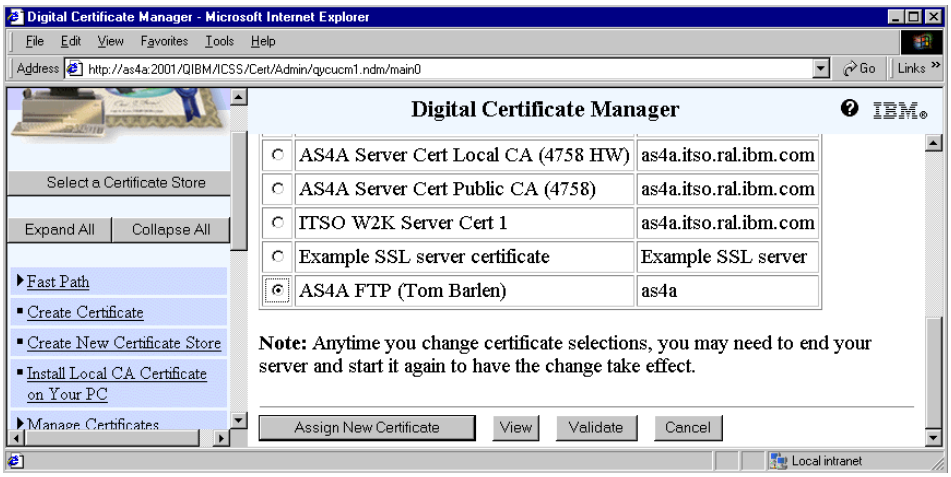


Figure 247. Digital Certificate Manager: Select a certificate

8. Select the certificate from the list that you want the FTP server to use to establish SSL connections between the FTP server and client.
9. Click **Assign New Certificate**.

   DCM displays a confirmation message as shown in Figure 248.

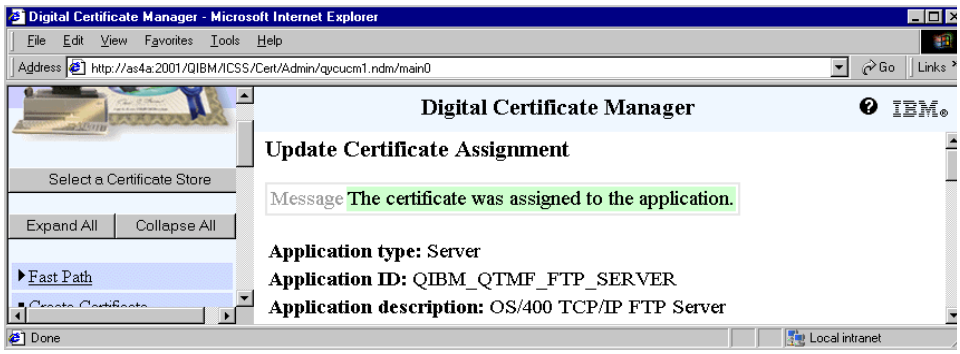


Figure 248. Digital Certificate Manager: Confirmation message

10. Click **Cancel** to return to the list of server applications.

5.3.2.3 Updating the application definition for client authentication

It is important to understand that when client authentication is set to Required in the FTP server application definition, each client that wants to connect via a secure connection must present a certificate that is associated with an OS/400 user profile.

Perform the following steps to enable client authentication for the FTP server:

1. Start DCM.
2. From the DCM navigation pane, click **Manage Applications**.
3. From the Manage Applications group, click **Update application definition**.
4. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed as shown in Figure 249.

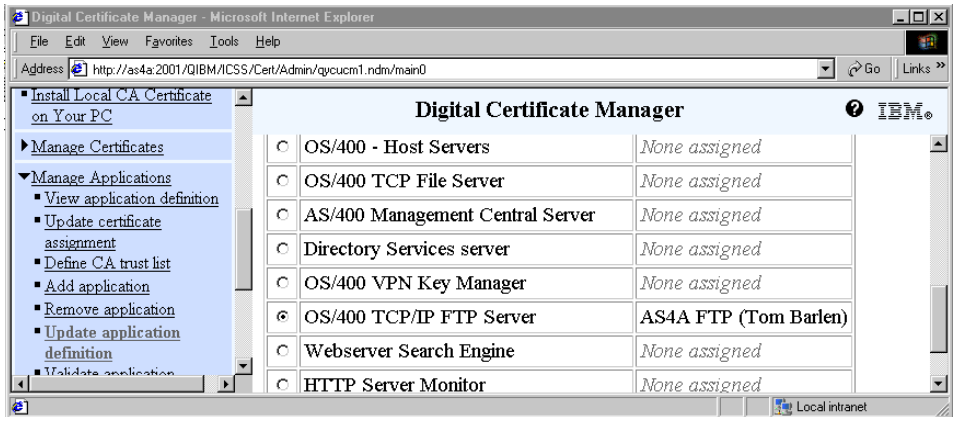


Figure 249. Digital Certificate Manager: Update Application Definition window

5. Select the **OS/400 TCP/IP FTP Server** and click **Update Application Definition**.

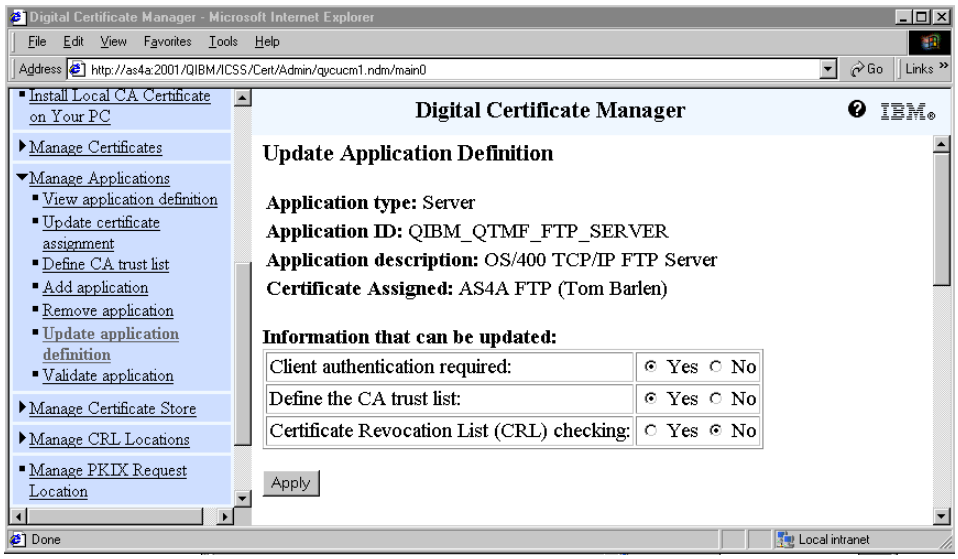


Figure 250. Digital Certificate Manager: Selecting client authentication

Mostly, when referring to enabling the FTP server for client authentication, it is meant to configure the server to require client authentication. The FTP server application itself is always enabled for client authentication, it just does not require it. As mentioned before, when you configure the server so that clients must authenticate themselves by presenting a certificate, clients without a valid certificate cannot establish a secure connection. Be careful when configuring the FTP server for required client authentication.

6. Set **Client authentication required** to **Yes** and click **Apply**.
7. A DCM confirmation message is displayed. Click **Cancel** to return to the list of server applications.

5.3.2.4 Updating the CA trust list

In this last task of the server setup, you need to specify which client certificates the FTP server accepts for client authentication. That is, the client certificates must be issued by a certificate authority that the FTP server trusts.

1. From the Manage Applications group click **Define CA trust list**.
2. As the application type, select **Server** and click **Continue**. The list of applications registered in DCM is displayed.
3. Select the **OS/400 TCP/IP FTP Server** and click **Define CA Trust List**.

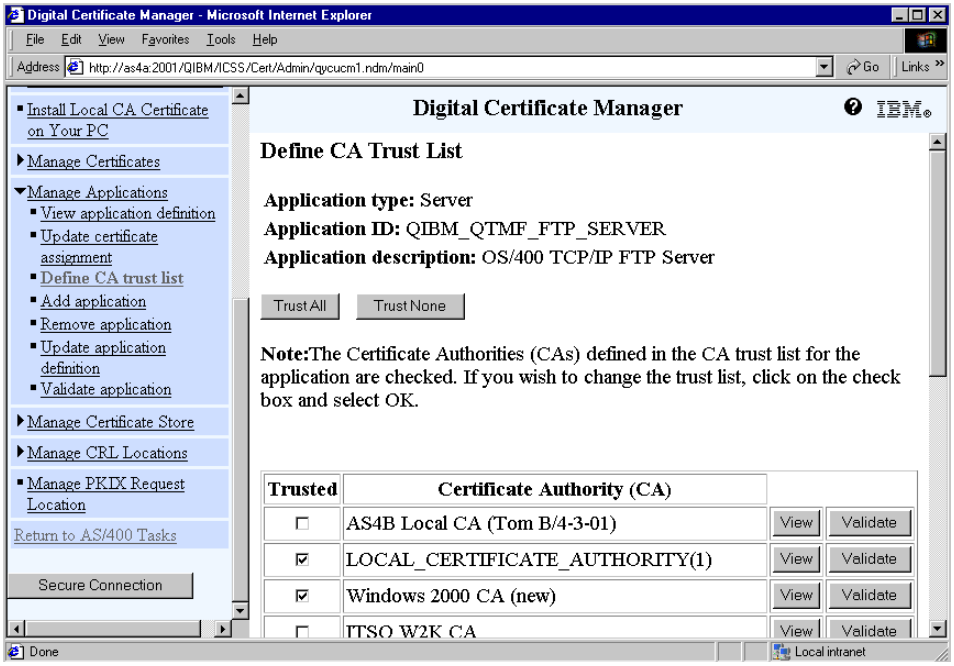


Figure 251. Digital Certificate Manager: Define CA Trust List

In the Define CA Trust List window, define the client or user certificates that this server application accepts for client authentication. In other words, only client or user certificates that are issued by a trusted CA are accepted by the server application. The list of displayed CAs contains only those CAs that are *Enabled* in the storewide CA list as defined in the Work with CA certificates option from the Fast Path menu. This means, if you installed a CA certificate, and it is not shown on the Define CA Trust List window, you must ensure that the CA is enabled.

4. Select all CAs that issued user or client certificates you want the FTP server to accept. Note that these certificates still need to be associated with an OS/400 user profile.
5. Click **OK**. The list of CAs is displayed again with a confirmation message stating that the CAs have been trusted.
6. Click **Done** to return to the list of server applications.
7. Restart the FTP server.

   To activate these changes to the FTP server, you must restart it using the following commands:

- To stop: `ENDTCPSVR SERVER(*FTP)`
- To start: `STRTCPSVR SERVER(*FTP)`

5.3.3 Configuring the PC

First, you must obtain a user or client certificate for the FTP client. Every time the client establishes a connection the certificate must be presented to the OS/400 TCP/IP FTP server.

Next, you must import the private CA certificate into FTP client. Note that if you use server and user certificates issued by a well-known CA, you do not need to exchange CA certificates. They are usually preloaded on all standard software applications. After the CA certificate is imported, you have to configure the client to use SSL.

5.3.3.1 Obtaining a user certificate and CA certificate

In this scenario we use a local OS/400 CA to create a user certificate. The certificate will then automatically be associated with the user profile that signed on to DCM. You could also obtain a certificate from a well-known CA and manually associate the certificate using DCM to a particular OS/400 user profile. The following steps show you how to create a user certificate:

1. Start DCM.
2. When prompted, sign on with a user profile you want to create the user certificate for. In this scenario we used a user profile without any special authorities.
3. From the DCM navigation pane, click **Create certificate**.

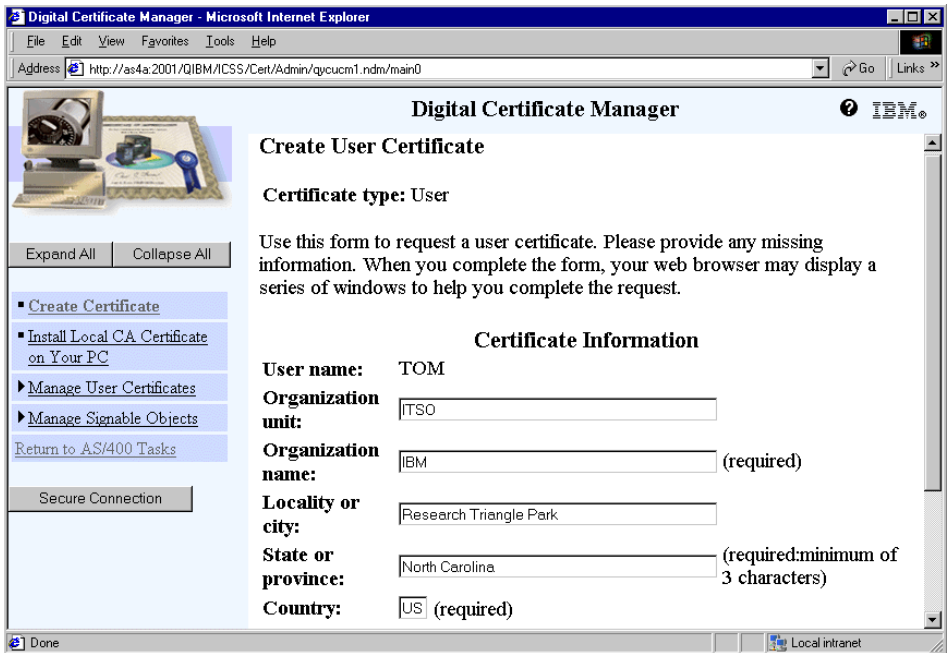


Figure 252. Digital Certificate Manager: Create User Certificate

4. Enter the data for the certificate's distinguished name. Note that the user name is already filled in and cannot be altered. This assures that the new certificate will automatically be associated with this user profile.
5. Click **Continue**. After the certificate is created, the following window is displayed. Note that the public/private key pair is generated in the PC's Web browser.

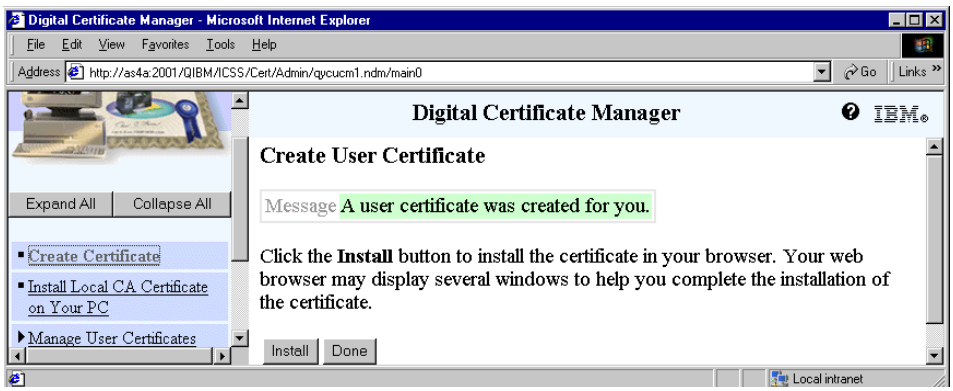


Figure 253. Digital Certificate Manager: Create User Certificate completion message

6. Click **Install** to install the signed user certificate into the browser's key database. A confirmation message is displayed. Click **OK** to continue. Note the messages vary depending on the browser version used to create the certificate.
7. Click **Done** to complete certificate creation process.

The user certificate is created and installed in browser's key database. To use the new user certificate with the FTP client, you need to export it from the browser's key database. In this scenario, we used the Microsoft Internet Explorer 5.01.

8. From the browser window's action bar, click **Tools->Internet Options**.
9. Click the **Content** tab.
10. Click **Certificates**.

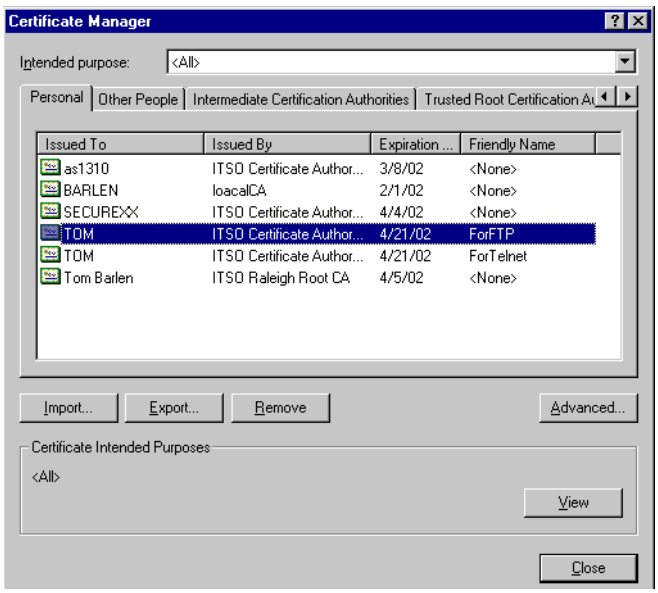


Figure 254. Internet Explorer: Certificate Manager

11. Select the user certificate that you just created and click **Export**. A Certificate Manager Export Wizard starts.
12. On the first window of the wizard, click **Next**.

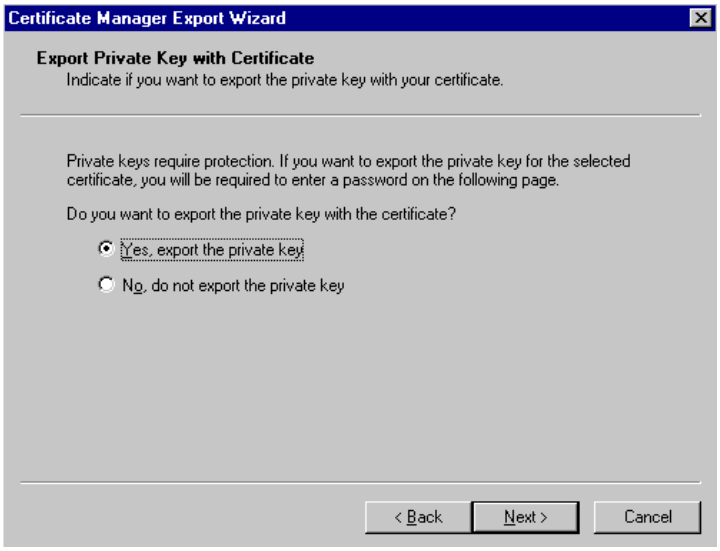


Figure 255. Internet Explorer: Certificate Manager Export Wizard

13. Select **Yes, export the private key** and click **Next**.

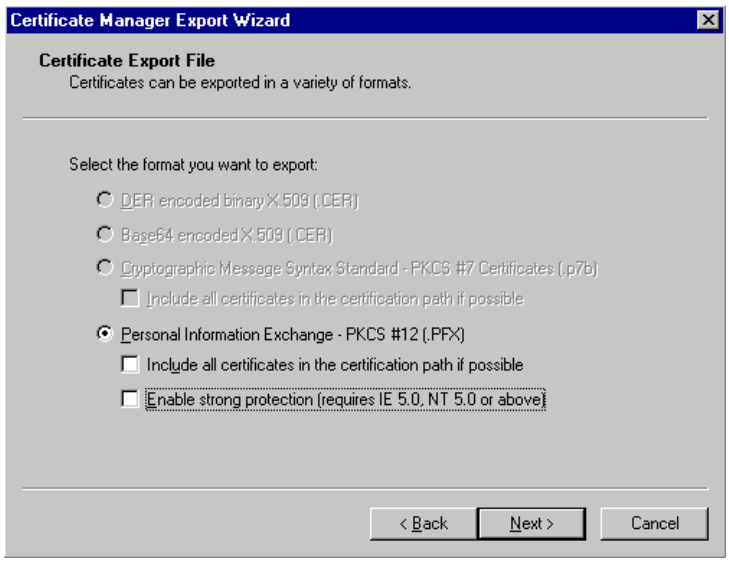


Figure 256. Internet Explorer: Certificate Manager Export Wizard

14. Deselect the **Enable strong protection (requires IE 5.0, NT 5.0 or above)** checkbox and click **Next**.

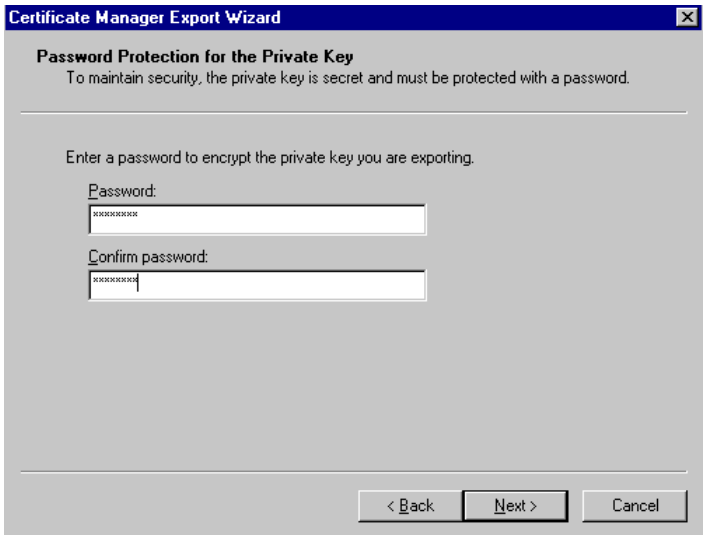


Figure 257. Internet Explorer: Certificate Manager

15. Enter a password. This password is used to encrypt the exported certificate including the private key. You need this password later when establishing a secure connection with the FTP client.
16. Click **Next**.
17. Enter a path and file name to store the exported certificate and click **Next**.

    In this scenario the following path and file name was used:

    `C:\xxx\tomftp.pfx`
18. Click **Finish** to compete the certificate export. A confirmation message is displayed. Click **OK** to close the message window.
19. Click **Close** to close the Certificate Manager and then **OK** to close the Internet Options window.

After the client (user) certificate has been exported into a file, you also need to make the CA certificate available for the FTP client to use.

5.3.3.2 Preparing the local CA certificate for the FTP client

The steps documented in this section show you how to prepare the CA certificate that was used to issue the server and client (user) certificate for this scenario.

Note

At the time this Redbook was written, the BlueZone Secure FTP client still required the Seagull Software Security Server product to prepare the CA certificate to be accepted as a trusted CA by the client. Seagull Software, however, told us that they will include a CA certificate import function in one of their new client versions. Therefore, the steps to trust a specific CA certificate by the client may change in the near future. Visit the Seagull Software Web site at `http://www.seagullsw.com` for updated versions of the FTP client.

The client version used in this scenario required the CA certificate to be hashed by the Security Server product. The file containing the CA certificate remains the same, but the file name will be changed to the hash generated by the Security Server. This section shows only the steps to generate the hashed certificate. For information about how to install and set up the Security Server product, refer to the Seagull Software Web site at `http://www.seagullsw.com`

1. From the PC where the Security Server is installed, start an FTP session to the iSeries or AS/400 server. You can use the FTP client provided with the Windows operating system.
2. Enter the following FTP commands after logging in to the FTP server to receive the local CA certificate from the iSeries or AS/400 server:

```
ascii
quote site nam 1
get /qibm/userdata/icss/cert/certauth/ca.txt  \xxx\ca.txt
quit
```

3. Start the Seagull Software Security Server.

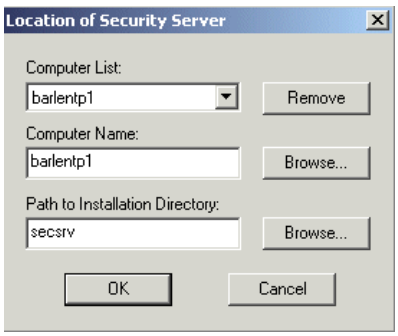


Figure 258. Location of Security Server

The Security Server can be managed by a management client. When starting the Security Server, you have to specify the computer name and the shared directory path where the Security Server product is installed.

4. Click **OK**. Depending upon whether you start the server the first time or you did not enter the license key before, you are prompted with the following message:

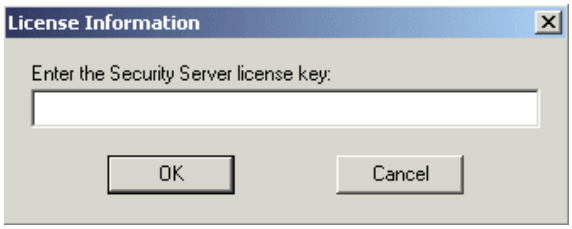


Figure 259. License Information

5. Enter the required information and click **OK**.

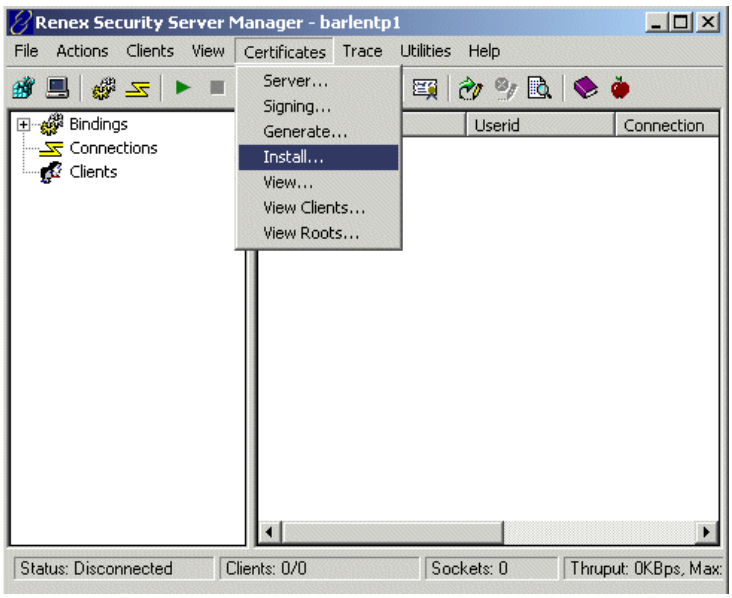


Figure 260. Security Server Main window

6. From the main window action bar, select **Certificates->Install**. The window as shown in Figure 261 on page 326 is displayed.

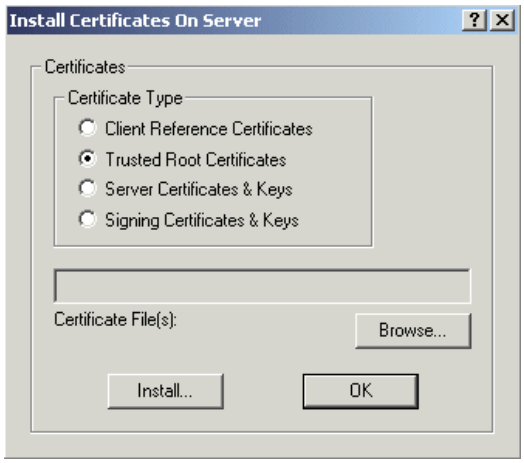


Figure 261. Install Certificates On Server window

7. Select **Trusted Root Certificates** and click **Browse**. Select the directory you received the CA certificate in and select the file containing the CA certificate. Then click the button displayed to complete the file selection. With Windows 2000 you need to click **Open**. You return to the window as shown in Figure 261, but this time the Certificate File(s) parameter is filled in.
8. Click **Install**. The hashed value is generated and the CA certificate file with the new name stored in the Security Server's directory as shown in Figure 262.

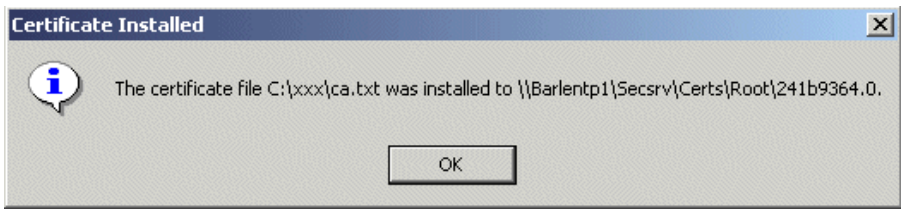


Figure 262. Certificate Installed window

9. Write down the name of the new file, in this case `241b9364.0`.
10. Transfer the file to the PC where the FTP client is installed and store it in the client's Certs directory. In this scenario it is the directory C:\BlueZone\Certs directory. Do not change the name of the file.

The local CA certificate is now trusted by the BlueZone Secure FTP client. Remember, the steps to establish the CA trust may change in a future version of the client software.

5.3.3.3 Configuring the FTP client for SSL with client authentication

In this scenario we used the BlueZone Secure FTP client Version 2.11 from the Seagull Software. You can find more information about this client at `http://www.seagullsw.com`. This section describes only the most important setup steps and is not intended to replace the product documentation. It is assumed that the client software is already installed.

1. Start the BlueZone Secure FTP client.
2. From the action bar, select **Session** as shown in Figure 263.

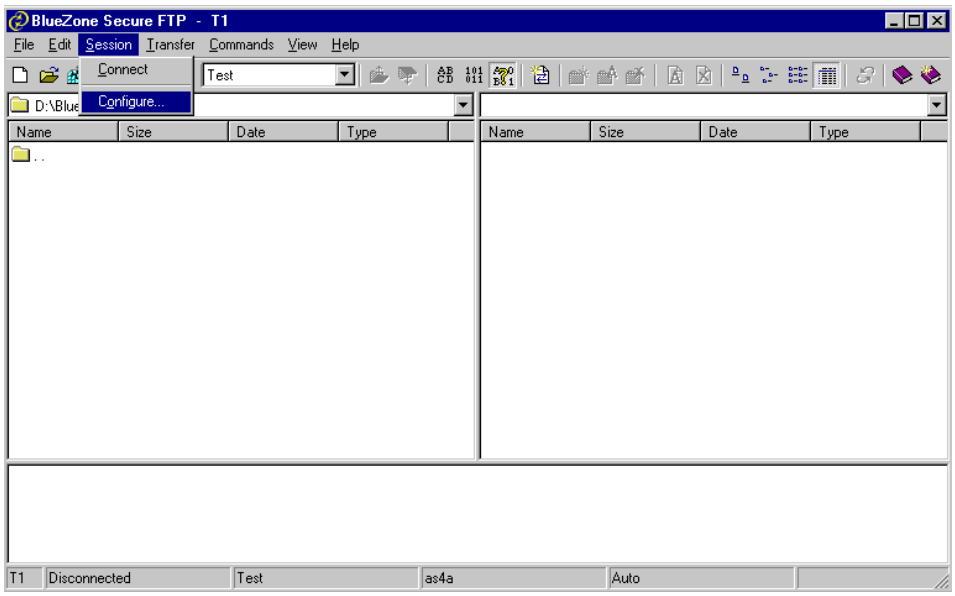


Figure 263. BlueZone Secure FTP client window

3. Select **Configure** from the Session menu. The window as shown in Figure 264 appears.

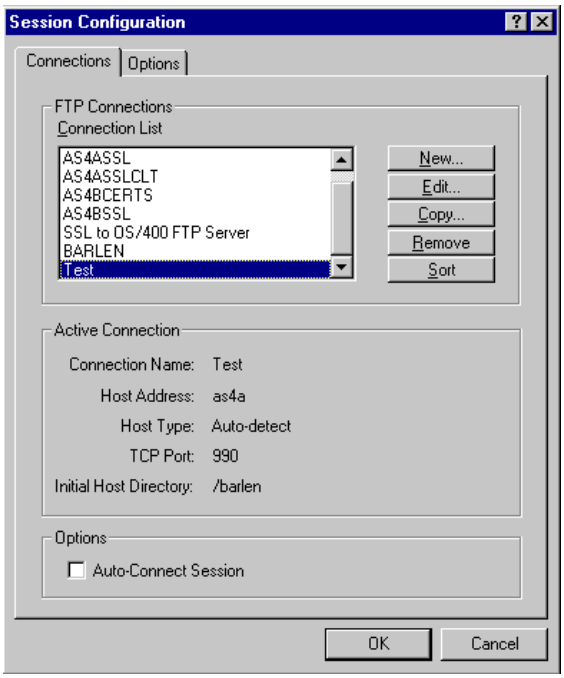


Figure 264. Session Configuration window

4. Click **New** to add a new configuration to the client. The window shown in Figure 265 appears. Click the **Connection** tab.

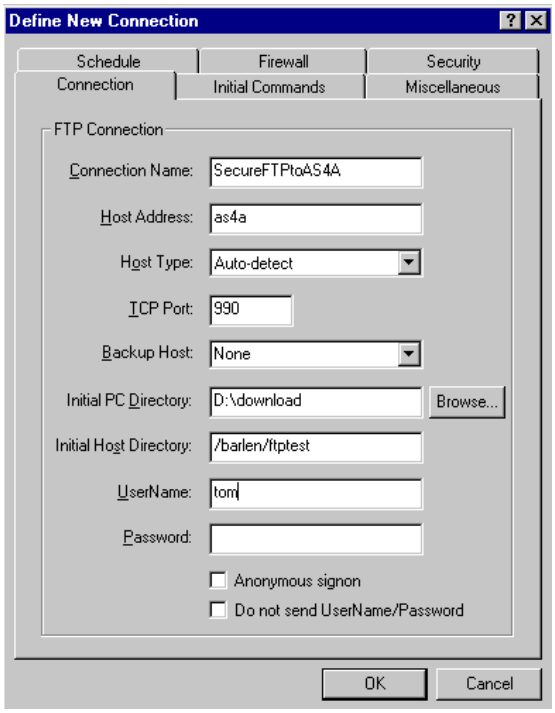


Figure 265. Define New Connection window: Connection tab

5. Enter the values for the connection properties:

   Connection name: Specify a name that identifies this particular configuration.

   Host Address: Specify the host name or IP address of the FTP server.

   Host Type: You can leave the default value or select IBM AS/400.

   TCP Port: Enter the port the server listens on for secure FTP connection requests. In this case, the OS/400 TCP/IP FTP Server listens on port 990.

   Backup Host: Specifies another client's configuration in case the host defined in this configuration is not accessible.

   Initial PC Directory: Defines the default PC directory that will be displayed in the left-hand pane of the client when establishing a connection.

Initial Host Directory: Defines the default OS/400 directory that will be displayed in the right-hand pane of the client when establishing a connection.

User Name: Enter the OS/400 user profile you created the user certificate for. Remember, the user certificate has to be associated with this user profile.

Password: You can leave this value blank when using a secure session with client authentication. The OS/400 TCP/IP FTP Server logs in the user automatically when the presented client (user) certificate is associated with the user profile specified in the User Name parameter.

Do not select the remaining checkboxes.

6. Click the **Security** tab (Figure 266).

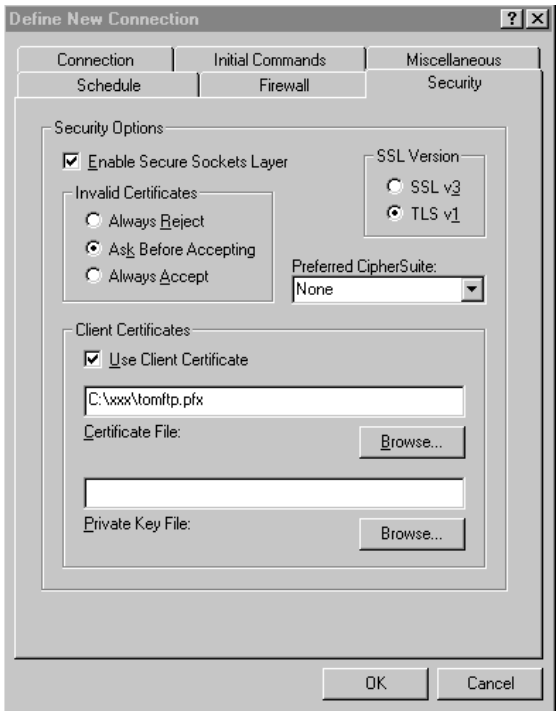


Figure 266. Define New Connection window: Security tab

7. Enter the values for the security properties:
   a. Select **Enable Secure Sockets Layer.**

b. You can select either **TLS v1** or **SSL v3**. Both protocols are supported by the OS/400 TCP/IP FTP Server. However, the TLS protocol is newer and solves some security weaknesses discovered in SSL v3. Thus, you are advised to select TLS v1 when the FTP server supports it.

c. The section titled Invalid Certificates specifies how to treat server certificates that are received during an SSL handshake if the CA that issued the server certificate is not in the client's list of trusted CAs. You may want to select **Ask Before Accepting**. Using this selection, a warning message is displayed when an untrusted server certificate is received.

d. By default, the Invalid Certificates parameter is set to Always Accept. If you are using this value, you do not require the CA certificate in the client's Certs directory. However, it is not recommended that you trust any CA.

e. Leave the default value for the Preferred CipherSuite parameter.

f. Select **Use Client Certificate**. If you want to use the client without client authentication, clear this parameter and configure the FTP server to not require a client certificate for authentication. In this case you also have to provide a password in the connection properties.

g. For the Certificate File parameter, enter the path and file name that contains the exported client (user) certificate. In this scenario it is `C:\xxx\tomftp.pfx`.

h. Note that the certificate used in this scenario is stored in PKCS#12 encoded format. That is, the file contains the certificate and the corresponding private key encrypted by using an encryption algorithm and a password.

i. Do not specify any value in the Private Key File parameter unless you are using a certificate that has its private key stored in a separate file.

8. Click **OK** to save the new configuration.

9. Click **OK** again to close the Session Configuration window.

5.3.4 Verifying the SSL connection

The following procedures can be used to determine if the FTP connection is protected using the SSL protocol.

On the iSeries server:

1. Use the `NETSTAT *CNN` command and check that the TCP port 990 and port 21 are in the Listen state.

```
Remote           Remote  Local
Address           Port    Port  Idle Time  State
*                    *     21  000:35:54 Listen
*                    *    992  000:03:13 Listen
```

2. When an FTP session is established, use the `NETSTAT *CNN` command to check that the FTP client has a connection through port 990 or 21. The ports depend on the client product. The BlueZone Secure FTP client establishes a connection over port 990. In addition to the control session you will also see a data session. In this case it is a passive FTP session with a dynamically assigned port.

```
Remote           Remote  Local
Address           Port    Port  Idle Time  State
10.10.62.54        1479    990  000:00:00  Established
10.10.62.54        1480  27184  000:00:00  Established
```

On the PC:

1. Start an FTP session that is configured for SSL. The BlueZone Secure FTP client prompts for the password that was used to protect the client certificate when a connection is established. It also logs the connection messages as shown in Figure 267. The following message indicates that the user was successfully authenticated by its presented certificate:

   ```
   232 User TOM authenticated via client certificate.
   ```

   This means the certificate presented during SSL handshake is associated with the user profile specified in the client configuration.

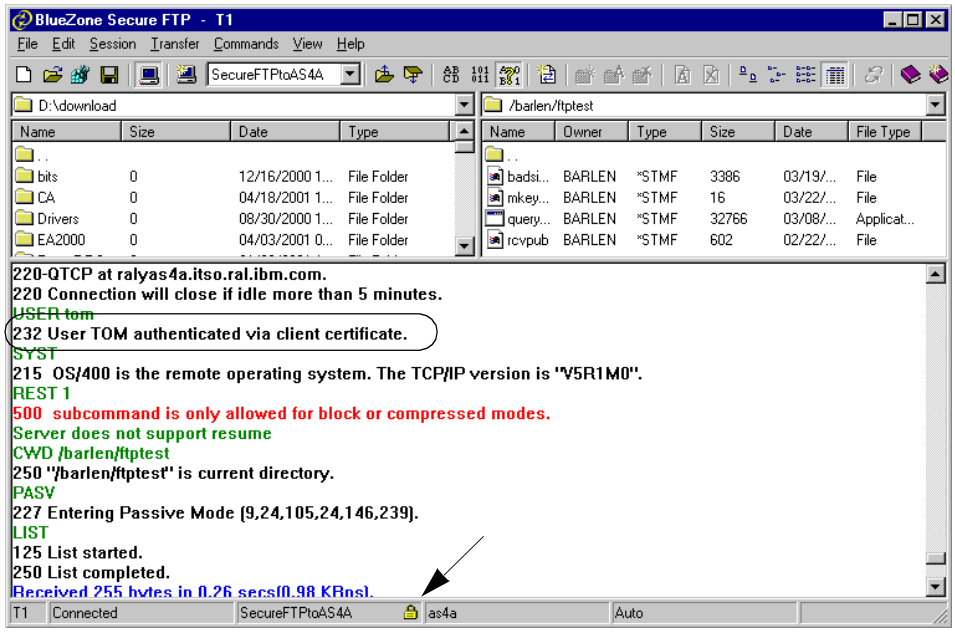


Figure 267. BlueZone Secure FTP client with active connection

The padlock in the status bar of the client indicates that this is a secure session. When you click on the padlock, another window opens with details about the server certificate and the cipher suite used for this connection.

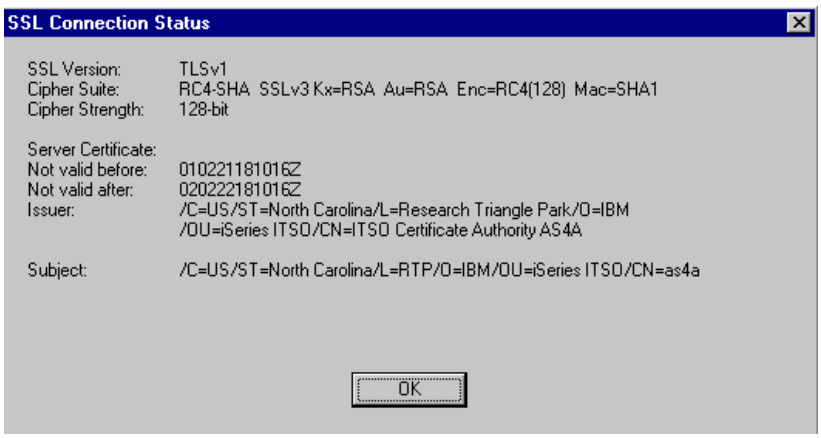


Figure 268. SSL Connection Status

Chapter 6. Using SSL in ILE RPG sockets applications

Although most sockets and Secure Socket Layer (SSL) applications are written in C, it is quite feasible to program sockets and SSL applications in ILE RPG. This may be a good choice if you are familiar with RPG and not with C, or if you are modifying an existing RPG application, or if you do not have access to a C compiler. All the SSL and sockets application programming interfaces (APIs) and all C functions can be used in ILE RPG on an iSeries or AS/400 system.

6.1 Programming techniques for using APIs and C functions in ILE RPG

Almost all C functions and all APIs are either procedures in service programs or independent programs. ILE RPG has several ways of calling external routines such as programs or procedures in service programs. Calls can be made without any preamble, by using the `CALL`, or `CALLB`, operation codes. However it is recommended, and sometime necessary, that you first define a prototype and then use the `CALLP` operation code, or simply name the procedure in an `EVAL` statement. Naming the procedure in an `EVAL` statement is the most common and convenient method. It also allows procedures to return a value - as many C functions do. For more information on calling procedures from ILE RPG, read the *ILE RPG Programmer's Guide* or *ILE RPG Reference* found in the iSeries Information Center by clicking **Programming->RPG->ILE**.

6.1.1 Prototypes

Briefly, a prototype is a description of a procedure. It documents the types of parameters the procedure uses and the value, if any, that the procedure returns. Consider the example in Figure 269.

```
 * Prototype for QExample, which............
D Example         pr           10i 0 extproc('QsyExample')
D $C1                          15
D $N1                          10i 0
                      .......
                      .......
 * Call example procedure to get result
C            Eval   Result = Example(25: 'ABC')
                      .......
```

Figure 269. Prototype example

Prototypes must be placed in the data definitions. This prototype defines the external procedure QsyExample as returning a 10-digit integer and requiring two parameters. Within the RPG program the name "Example" will be used to call the procedure.

Prototypes have several advantages:

- They allow the compiler to verify that the parameters used on your procedure calls are of the correct type and size.
- They allow you to call a procedure by name within an `EVAL` statement.
- They allow you to use constants on procedure calls such as 123 or "ABC". The compiler automatically converts these to the appropriate data types.
- They are the only way to obtain procedure return values, as in the example in Figure 269.

For more information on prototypes in ILE RPG programs, read the *ILE RPG Programmer's Guide* or *ILE RPG Reference* found in the iSeries Information Center by clicking **Programming->RPG->ILE**.

When programming in C, the prototypes for all the C functions, the associated data structures, and useful constants are readily available. They are held in files such as QSYSINC/H member STDIO. C program source normally includes instructions for the relevant files to be copied in by the compiler. Unfortunately such files are not generally available for ILE RPG. This can make getting started somewhat slow, since you have to define the prototypes and data structures needed for the APIs and C functions you want to use. However we have created prototype files for ILE RPG for all the sockets APIs, all the object signing APIs, many of the APIs for working with certificates and validations lists, and a few other useful APIs we needed. Instructions for obtaining these are in Appendix G, “Using the additional material” on page 471.

When defining prototypes and their associated data structures for use in ILE RPG, it is best to put them in a separate file of related functions. In your programs you can use `/copy file-name,member-name` at the start of the data definition section so that the definitions are copied into your source by the compiler. This keeps your definitions in one place, ensuring consistency and saving retyping.

6.1.2 Defining prototype parameters

In ILE RPG parameters can be passed to external procedures by reference or by value. Parameters can be of three basic types, a variable, a fixed-length structure, or something without a fixed length.

When setting up a prototype for a C function, or an API written for C, you must be very careful about which parameters are pointers.

6.1.2.1 Passing parameters by reference

When a parameter is passed by reference, a pointer to the parameter is given to the called procedure. The procedure uses this to access the original data. It is very important that the calling and called procedures agree on the data type and length; otherwise strange and elusive bugs can occur in your program. APIs implemented as individual programs pass all parameters by reference. You can recognize these by their names, which will be something like *QSYDRGAP*, instead of *QsyDeregisterAppForCertUse*. In the API documentation the text will state if the API is *OPM*, Original Program Model, that is an individual program, or *ILE*, Integrated Language Environment, that is a procedure in a service program. Where an API is implemented in both ways, the parameters will still all be passed by reference.

6.1.2.2 Passing parameters by value

When a parameter is passed by value, a copy of its contents is copied to the procedure. The procedure cannot alter the original data. C procedures always pass parameters by value. However, often the parameters are pointers and passing a pointer (to variable XYZ) by value is almost the same as passing variable XYZ by reference. The only difference is that when passing by reference the parameter is defined in the prototype and the compiler can check that the correct type of variable is passed. When passing a pointer the compiler can only check that a pointer is given. It cannot check what it points to.

6.1.2.3 Passing variables

Variables are easy. Just define them as described in the API or C function documentation. For example, the definition of the Socket API in the C prototype and in our prototype for ILE RPG, as shown in Figure 270.

```
int socket(int address_family, int type,
             int protocol)

 * Prototype for SOCKET which creates a socket description for later use.
D Socket          pr            10i 0 extproc('socket')
D $AddrFamily                   10i 0 value       Address family
D $CommType                     10i 0 value       Comms type
D $Protocol                     10i 0 value       Protocol used
```

Figure 270. Socket API definition and prototype

The RPG equivalents for the OS/400 C numeric type are:

int 10-digit integer, 10i 0

u_int 10-digit unsigned, 10u 0

short 5-digit integer, 5i 0

u_short 5-digit unsigned, 5u 0

long_int 10-digit integer, 10i 0

u_long_int 10-digit unsigned, 10u 0

others See the C language documentation in the *ILE C/C++ Language Reference* found in the iSeries Information Center by clicking **Programming->C and C++->ILE**.

6.1.2.4 Passing fixed-length structures

Fixed-length structures should be defined by first defining the structure, then defining a parameter like the structure. Consider the example in Figure 271.

```
 * Address structure used for sockets and IP type connections
D#IP_Adr_Str      ds
D #IP_Adr_Family             10i 0              Address family
D #IP_Adr_Port               10i 0              Port number
D #IP_Adr_Adr                 4                 4 byte IP addr
 *                                              in hexadecimal
D #IP_Adr_Res                 4                 Reserved

 * Prototype for XYZ utility which......
D XYZ             pr                extproc('XYZUtil')
D $IP_Adr_Str                       like(#IP_Adr_Str)
D $Count                      10i 0
```

Figure 271. Example of using a structure in a prototype

If you define prototypes in a file that is automatically copied into your program source, the structures defined there can also be used in your programs. You should use a naming convention to avoid confusion and accidentally reusing names.

If you have structures that you want to use several times within a program, or if you do not want to allocate storage to structures until they are needed, you could use basing pointers. To find how to do this, read the relevant parts of the *ILE RPG Programmer's Guide* found in the iSeries Information Center by clicking **Programming->RPG->ILE->ILE RPG Programmer's Guide**.

6.1.2.5 Variable-length parameters

Variable-length parameters can be handled in two ways. Both pass a pointer to the parameter. You can do this by defining the parameter in the prototype as a pointer passed by value, or as a fixed parameter passed by reference adding the keyword `OPTIONS(*VARSIZE)`. For example:

```
 * Prototype for XYZ utility which......
D XYZ            pr                 extproc('XYZUtil')
D $DomainName                   *   value        Null term string
D $MsgDta                  32767    options(*varsize)
D $MsgDtaLenL                 10i 0
D $Count                      10i 0
                        ......
                        ......
* Your RPG program
D Msgtext                    160
D Len                         10i 0
                        ......
C          Eval     Len=%len(%trim(MsgText))
C          Eval     XYZ(%addr(Domain): MsgText: Len: CallCount)
```

Figure 272. Example of using a variable-length parameter in a prototype

Since $DomainName is defined as a pointer, we must pass a pointer data type. `%Addr(Domain)` returns a pointer to the variable Domain, and the value of this is passed to procedure XYZUtil, which uses it to access the data in the variable. $MsgDta is defined as a character field; because of `options(*varsize)`, fields smaller or larger fields than 32767 bytes may be passed. We pass MsgText, which is 160 bytes long.

Variable-length parameters are quite common, for example C character strings, which use x'00' to mark the end of the string. Many of the structures returned by APIs include variable-length lists.

You should use the technique that feels most appropriate. For example, the API for sending program messages, QMHSNDPM, has a parameter for the message data that can be up to 32767 bytes long followed by a parameter to specify the length of the message data. It would be best to declare the message data parameter a character field of varying size. Another example is the sockets API *recv*. This has a parameter for the buffer to hold the data received. The C documentation declares this as a pointer and it is best to follow this definition. Declaring as a pointer also allows you to work with offsets. Look at `%addr(Buffer)+TotSent` in the example shown in Figure 273 on page 340.

```
 * Loop receiving data echoed back from the server
C        Eval      TotSent = 0
C        Eval      Buffer = *blank
C        DoW       TotSent < BufferLen
C        Eval      BytesSent = Recv(SocDesc:
C                                  %addr(Buffer)+TotSent:
C                                  BufferLen-TotSent:
C                                  RecvFlags)
C        Eval      TotSent = TotSent + BytesSent
C        EndDo
```

Figure 273. Using a pointer with an offset

6.1.2.6 Pointer parameters in C procedure definitions

C functions and APIs written for C programming often pass pointers as parameters. You must determine what pointers point at and define your prototype parameters accordingly as described previously. Pointers are indicated by a * preceding the parameter name. For example, see Figure 274.

```
int getsockopt(int socket_descriptor,
               int level, int option_name,
               char *option_value, int *option_length)
```

Figure 274. Example of pointers used by an API

In this example, the function itself returns an integer. *socket_descriptor, level* and *option_name* are integers, **option_value* is a pointer to a character string, and **option_length* is a pointer to an integer. The first three parameters are defined in the prototype as integers passed by value. The character string is in a variable length so a pointer, passed by value, is defined in the prototype instead of a fixed-length variable. Option length is passed as a pointer and is defined in the prototype without the `value` keyword so that it is passed by reference, that is by a pointer. Hence the prototype appears as shown in Figure 275.

```
 * Prototype for GETSOCKOPT which returns various socket option settings
D GetSockOpt      pr           10i 0 extproc('getsockopt')
D $SocDesc                     10i 0 value            Socket descriptor
D $SocOptLvl                   10i 0 value            Option level
D $SocOption                   10i 0 value            Option name
D $SocOptVal                     *   value            Option value
D $SocOptLen                   10i 0                  Option value length
```

Figure 275. Example prototype for an API using pointers

6.2 Where to find API and C function documentation

All API documentation can be found in the Information Center by clicking **Programming->CL and APIs->APIs** then one of **Alphabetical list of APIs**, **APIs by category**, or **APIs by description**.

If you know the name or description of the API you want, the APIs by description option is good. You can use your browser's search function (Ctrl+F normally) to search the list for the API name or for words describing the API.

C functions are also documented in the Information Center. Read *ILE C for AS/400 Run-Time Library Reference* or *IBM Open Class Library Reference* found in the iSeries Information Center by clicking **Programming->C and C++->ILE**.

Functions intended for use in C programs will almost always specify the file, or files, holding the required prototype and data structure definitions. There will be one or more lines such as `#include <sys/socket.h>`, or `#include <stdio.h>`. The first case specifies the member `SOCKET` in file SYS. The second case specifies the member `STDIO` in the default file H. QSYSINC is the library holding these files.

There is also the file QRPGLESRC in library QSYSINC. This holds RPG definitions for data structures used by program APIs, such as QSYVLDL (which holds data structures used by validation list handling APIs). They can be useful for obtaining the data structures you need. However, they do not contain prototype definitions, nor definitions of constants used, such as return codes. We have created ILE RPG files defining the constants, data structures, and prototypes for the object signing APIs, many of the sockets APIs, all the SSL APIs, and a few other APIs we used along the way. This was a considerable task. If you will be programming in RPG using these APIs, we recommend you download and use these files. Instructions for obtaining them are in Appendix G, "Using the additional material" on page 471.

6.3 Programming with error codes returned from APIs

Most C functions, and APIs written for C, return error codes via a global variable. Some use other methods, such as the sockets API *send*, where you have to use another API *getsockopt* to retrieve the error reason code.

Most APIs written for general use return a pointer to a standard data structure.

6.3.1 Accessing the C error codes

The C global error variable can be accessed using the function *errno* which returns a pointer to the error code. The error code is an integer; however, the function documentation will talk of codes. For example “When recv() fails, errno can be set to one of the following: [EACCES] Permission denied. [EBADF] Descriptor not valid.” The codes EACCES and EBADF are constants defined along with prototypes and data structures so that you can use something meaningful in your program instead of a number. We have put all the error number definitions in our ILE RPG prototype definition file, adding the prefix @ to reduce the chance of duplicate names (C names are case sensitive and so the chance of duplicates is less). Unfortunately sometimes the C names were too long and had to be abbreviated, so you cannot always copy from C code. Figure 276 is an example of using the C global error code.

```
D ReasonPtr       s          *                    Err rsn pointer
D Reason          s        10i 0 based(ReasonPtr)  Err reason
 *
/copy qrpglesrc,proto_soc
                     ......
                     ......
C         Eval      BytesSent = Recv(SocDesc2:
C                                  %addr(Buffer)+TotSent:
C                                  MaxData: RecvFlags)
C         If        BytesSent < 0
C         Eval      ReasonPtr = errno()
 * If permission was denied....
C         If        Reason = @EACCES
```

Figure 276. Example of using the C global error code

Tip

To find the description of a C global error code, you can display the corresponding CPEnnnn message description. For example, when you receive an error code 3420, you can display the error code description by using the command `DSPMSGD CPE3420`, which displays `Address already in use`.

6.3.2 Using the standard error structure

The standard error structure can be set so errors result in an escape message being sent to the program or so the error details are made available in a data structure for the program to access.

For testing we found it best to have escape messages sent. We could then read the message in the joblog and display the second-level text if we wanted to. This is good for debugging. However, if you want your programs to handle error conditions you will need to use the error structure.

We obtained the error structure definition by copying in `QSYSINC/QRPGLESRC` member `QUSEC`. This is a very basic structure as shown in Figure 277.

```
DQUSEC            DS
D*                                          QUSEC
D QUSBPRV                1      4B 0
D*                                          Bytes Provided
D QUSBAVL                5      8B 0
D*                                          Bytes Available
D QUSEI                  9     15
D*                                          Exception Id
D QUSERVED              16     16
```

Figure 277. QUSEC standard error structure

If you want to have an escape message when an error occurs, set bytes provided to zero. If you just want to access the message ID in your program, set bytes provided to 16 and check for the presence of a message ID. If you want message data, define a field of appropriate length following QUSERVED and set bytes provided to the total length of the structure. For more information about using the error structure read *API Error Reporting* found in iSeries Information Center by clicking **Programming->CL and APIs>OS/400 concepts**.

6.4 Overview of programming sockets and SSL applications

This section discusses the flow of control between server and client, and the APIs used in a very basic SSL application. For a much more complete discussion of sockets programming and for examples, see *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming**.

On the AS/400 system there are two sets of SSL APIs. We have used the newer Global Secure Toolkit APIs. The documentation states “They have more options and functionality than the SSL APIs”. They are also easier to program with than the older APIs.

Here we describe the flow and the APIs used for establishing an SSL connection and sending secured data. For sending unencrypted data, the flow and APIs are the same; just omit all the GSK APIs and use the APIs *send* and *recv* to send and receive data.

The following table summarizes the flow of control. Functions within a cell are not dependent on each other. Functions in sequential cells must be performed sequentially. Arrows in the middle column show where client and server interact.

Table 10. Sockets application flow

Server		**Client**
Prepare socket. APIs *socket, setsockopt, bind, listen* Prepare SSL environment. APIs *gsk_environment_open, gsk_attribute_set_buffer, gsk_attribute_set_enum, gsk_attribute_set_numeric-value, gsk_environment_init*		Prepare socket. APIs *socket, setsockopt*, *gethostbyname* Prepare SSL environment. APIs *gsk_environment_open, gsk_attribute_set_buffer, gsk_attribute_set_enum, gsk_attribute_set_numeric-value, gsk_environment_init*
Loop processing clients requests		
Accept client connections. API *accept*	←	Connect to server. API *connect*
Prepare SSL session. APIs *gsk_secure_soc_open, gsk_attribute_set_buffer, gsk_attribute_set_enum, gsk_attribute_set_numeric-value*		Prepare SSL session. APIs *gsk_secure_soc_open, gsk_attribute_set_buffer, gsk_attribute_set_enum, gsk_attribute_set_numeric-value*
Start SSL handshake. API *gsk_secure_soc_init*	↔	Start SSL handshake. API *gsk_secure_soc_init*
Check handshake result. APIs *gsk_attribute_get_buffer, gsk_attribute_get_enum, gsk_attribute_get_numeric-value, gsk_attribute_get_cert_info*		Check handshake result. APIs *gsk_attribute_get_buffer, gsk_attribute_get_enum, gsk_attribute_get_numeric-value, gsk_attribute_get_cert_info*

Server		Client
Communicate with client. APIs *gsk_secure_soc_read, gsk_secure_soc_write, send, recv, read, write*	◄──►	Communicate with server. APIs *gsk_secure_soc_read, gsk_secure_soc_write, send, recv, read, write*
Close SSL session. API *gsk_secure_soc_close* Close session socket. API *close*		Close SSL session. API *gsk_secure_soc_close* Close SSL environment. API *gsk_environment_close* Close socket. API *close*
Repeat loop for next client request		
Close SSL environment. API *gsk_environment_close* Close socket. API *close*		

6.4.1 The server application flow in more detail

1. The server sets up a socket on which it will receive requests.
   a. Create a socket. *socket* is used to create a socket of the required type. This returns a socket descriptor, which must be used in all future references to this socket.
   b. Allow the socket to be reusable, which improves performance if multiple requests are to be handled. *setsockopt* is used to set this attribute. It can also be used to set many other attributes.
   c. Bind to a local address. *bind* is used to associate the socket with a local address and port number.
   d. Set the socket to listen for incoming calls. *listen* is used to do this. If not done, incoming calls are ignored.
2. The server sets up the SSL environment, that is the various attributes that will be used to control SSL sessions with client applications. This can be done before or after setting up the socket. Some of the SSL attributes can be changed later for specific sessions.
   a. Create an SSL environment. *gsk_environment_open* is used to create an SSL environment. This returns a “handle”, that is a pointer, to the environment. The handle is used in all subsequent references to the SSL environment.
   b. Set the attributes of the environment. *gsk_attribute_set_buffer* is used to set character type attributes such as the application ID, or the cipher list to be used. *gsk_attribute_set_enum* is used to set attributes that can take only certain fixed values such as the session type. *gsk_attribute_set_numeric_value* is used to set attributes that can take

a range of numeric values, such as timeout periods. When all attributes are set, *gsk_environment_init* is used to initialize the environment.

3. The server accepts and processes incoming connection requests. Normally this would be in a loop that may be repeated many times.
   a. Accept calls. *accept* is used to wait for incoming calls. When a call is received this API returns a new socket descriptor created for the session. The original socket can continue to receive calls - up to a certain queue length, which is an option that can be specified on the *listen* API.
   b. Create an SSL session environment. *gsk_secure_soc_open* is used to create an SSL session using the SSL environment attributes set earlier. A handle to the session is returned that is used in all subsequent references to this SSL session.
   c. Associate the session socket with the SSL session. *gsk_attribute_set_numeric_value* is used to do this.
   d. Set session attributes. Session attributes may be changed if required using the APIs *gsk_attribute_set_buffer*, *gsk_attribute_set_enum*, and *gsk_attribute_set_numeric_value*.
   e. Negotiate the SSL handshake. *gsk_secure_soc_init* negotiates the SSL handshake finding, if possible, a set of attributes that both the server and client can use. Once complete, you have a secure connection.
   f. Check the SSL session attributes. For example, if client authentication is optional, the server will want to determine if a certificate was given. Or the server may want to check if the client certificate was signed by a trusted CA. The APIs *gsk_attribute_get_buffer*, *gsk_attribute_get_enum*, and *gsk_attribute_get_numeric_value* can be used to obtain the negotiated session details. The API *gsk_attribute_get_cert_info* can be used to obtain the other party's certificate so that you can store it or examine it.
4. The server and client exchange data. This can be done using both the SSL APIs, which encrypt the data, and the standard sockets APIs, which do not. However, if mixing modes, you must be very careful that sending types and receiving types match, mistakes can hang the connection.
   a. Send encrypted data using *gsk_secure_soc_write* or send unencrypted data using *send* or *write*, *send* is preferred.
   b. Read encrypted data using *gsk_secure_soc_read*, or read unencrypted data using *recv* or *read*. *recv* is preferred.
5. The server closes the session.
   a. Close the SSL session environment using *gsk_secure_soc_close*.

   b. Close the session socket using the *close*.

6. The server processes, or waits for, the next client.
7. The server closes the SSL environment and the socket on which it is receiving requests.
   a. Close the SSL environment using *gsk_environment_close*.
   b. Close the socket using *close*.

If the server must handle multiple client connections simultaneously, or if the total workload coming from clients is very high and requires multiple jobs in order to process it all, then additional techniques and APIs are required. See *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming** for more information and for example programs in C.

6.4.2 The client application flow in more detail

1. The client sets up a socket which it will use to contact the server.
   a. Create a socket. *socket* is used to create a socket of the required type. This returns a socket descriptor, which must be used in all future references to this socket.
   b. Find the IP address of the server if this is not known. *gethostbyname* can be used to find the IP address of a host name.
2. The client sets up the SSL environment, that is the various attributes that will be used to control the SSL session with the server. This can be done before or after setting up the socket. Some of the SSL session attributes can be changed later if required.
   a. Create an SSL environment. *gsk_environment_open* is used to create an SSL environment. This returns a *handle*, that is a pointer, to the environment. The handle is used in all subsequent references to this SSL environment.
   b. Set the attributes of the environment. *gsk_attribute_set_buffer* is used to set character type attributes such as the application ID, or the cipher list to be used. *gsk_attribute_set_enum* is used to set attributes that can take only certain fixed values such as the session type. *gsk_attribute_set_numeric_value* is used to set attributes that can take a range of numeric values such as timeout periods. When all attributes are set *gsk_environment_init* is used to initialize the environment.
   c. Create an SSL session environment. *gsk_secure_soc_open* is used to create an SSL session using the SSL environment attributes as

defaults. A handle to the session is returned that is used in all subsequent references to this SSL session.

3. The client calls the server and establishes a secure session.
   a. Connect to server. *connect* is used to establish a session with the server.
   b. Associate the session socket with the SSL session. *gsk_attribute_set_numeric_value* is used to do this.
   c. Set session attributes. Session attributes may be changed if required using the APIs *gsk_attribute_set_buffer*, *gsk_attribute_set_enum*, or *gsk_attribute_set_numeric_value*.
   d. Negotiate the SSL handshake. *gsk_secure_soc_init* negotiates the SSL handshake finding, if possible, a set of attributes that both the server and client can use. Once complete you have a secure connection.
   e. Check the SSL session attributes. For example you may want to check what cipher was negotiated or you may want to obtain the server's certificate so you can store it or examine it. The APIs *gsk_attribute_get_buffer*, *gsk_attribute_get_enum*, and *gsk_attribute_get_numeric_value* can be used to obtain the negotiated session details. The API *gsk_attribute_get_cert_info* can be used to obtain the server's certificate.
4. The server and client exchange data. This can be done using both the SSL APIs, which encrypt the data, and the standard sockets APIs, which do not. However, if mixing modes, you must be very careful that sending types and receiving types match; mistakes can hang the connection.
   a. Send encrypted data using *gsk_secure_soc_write*, or send unencrypted data using *send* or *write*. *send* is preferred.
   b. Read encrypted data using *gsk_secure_soc_read*, or read unencrypted data using *recv* or *read*. *recv* is preferred.
5. The client closes the session.
   a. Close the SSL session environment using *gsk_secure_soc_close*.
   b. Close the SSL environment using *gsk_environment_close*.
   c. Close the socket using *close*.

6.5 Adding an application description to the Digital Certificate Manager

This section discusses the steps and considerations required to add server and client application descriptions using DCM.

In order for SSL applications to use the Digital Certificate Manager support, they must be registered with the DCM. Your programs name the application and the DCM will retrieve the assigned certificate, perform certificate trust list checking, CRL checking, and so on. SSL applications can work without involving DCM; the APIs allow the key ring database, its password, and the required certificate to be named directly. However, it is simpler to use the DCM and much easier to manage the certificates involved. All our examples assume the DCM is being used.

6.5.1 Adding a server SSL application

Perform the following steps to add a secure application for certificate use through the DCM interface.

1. Via the AS/400 Tasks page, log on to the DCM.
2. Click **Select a certificate store.**
3. Select ***SYSTEM** and click **Continue.**

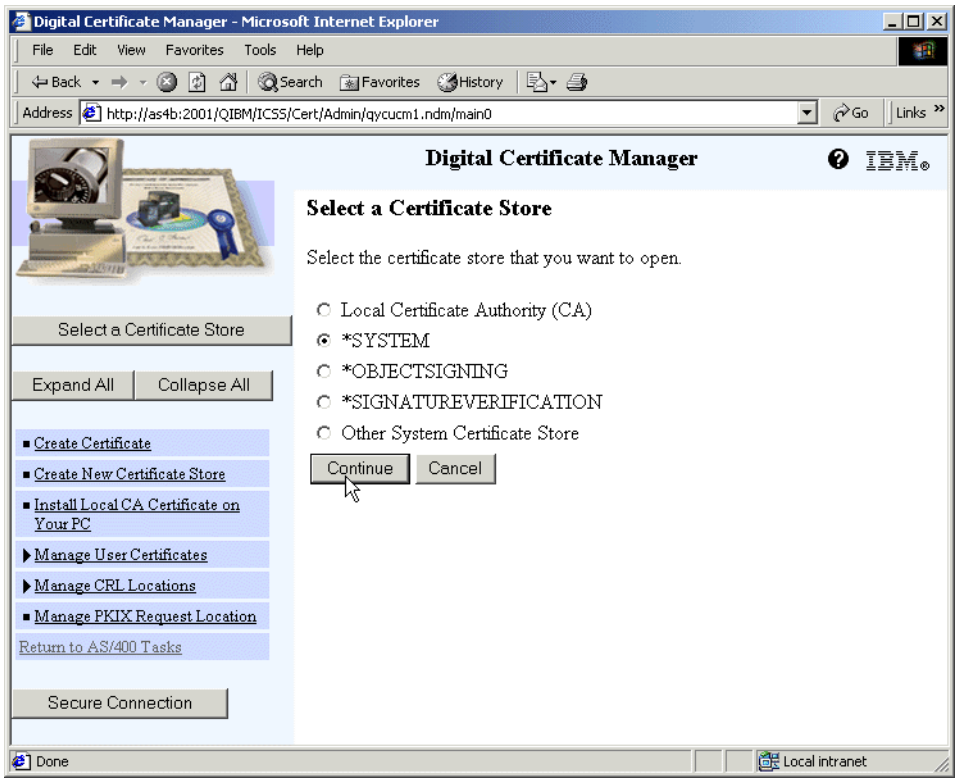


Figure 278. Select certificate store for your application

4. Type the certificate store password and click **Continue.** The menu in the navigation pane on the left refreshes showing the available options for the *SYSTEM certificate store.
5. Click **Fast Path.** Both panes refresh and the fast path options are displayed.

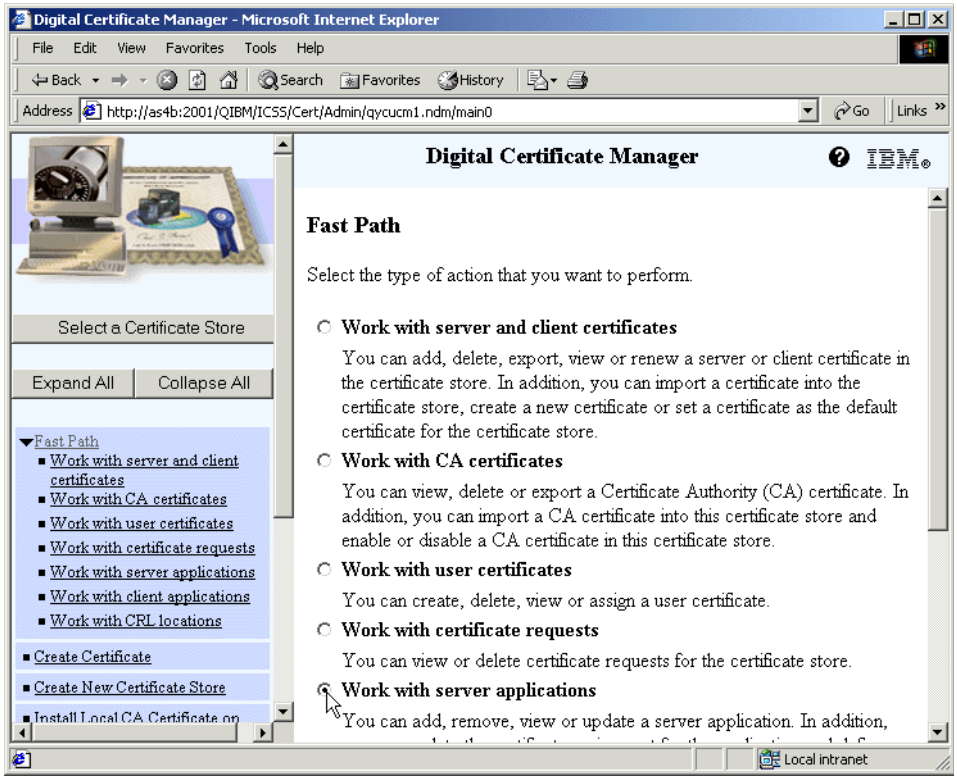


Figure 279. Fast Path options

6. Select **Work with server applications** and click **Continue** at the bottom of the window. You may need to scroll down to expose the **Continue** button.

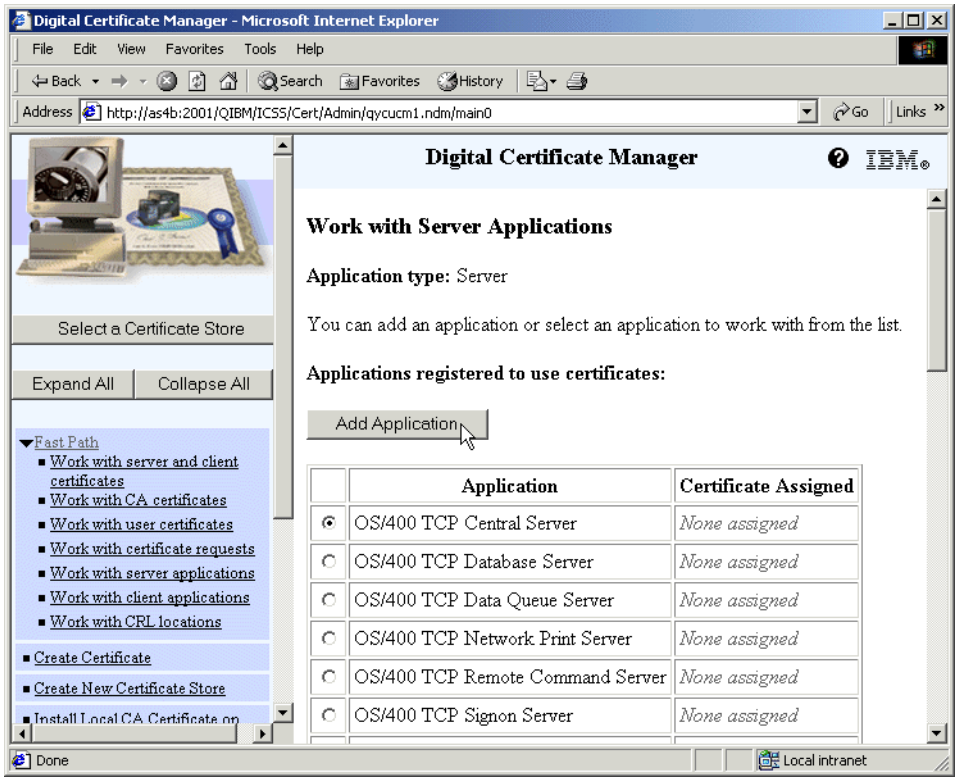


Figure 280. Add application

7. A list of all the existing server applications registered in the DCM appears. Click **Add application**. A window appears (see Figure 281) allowing you to enter the application details.

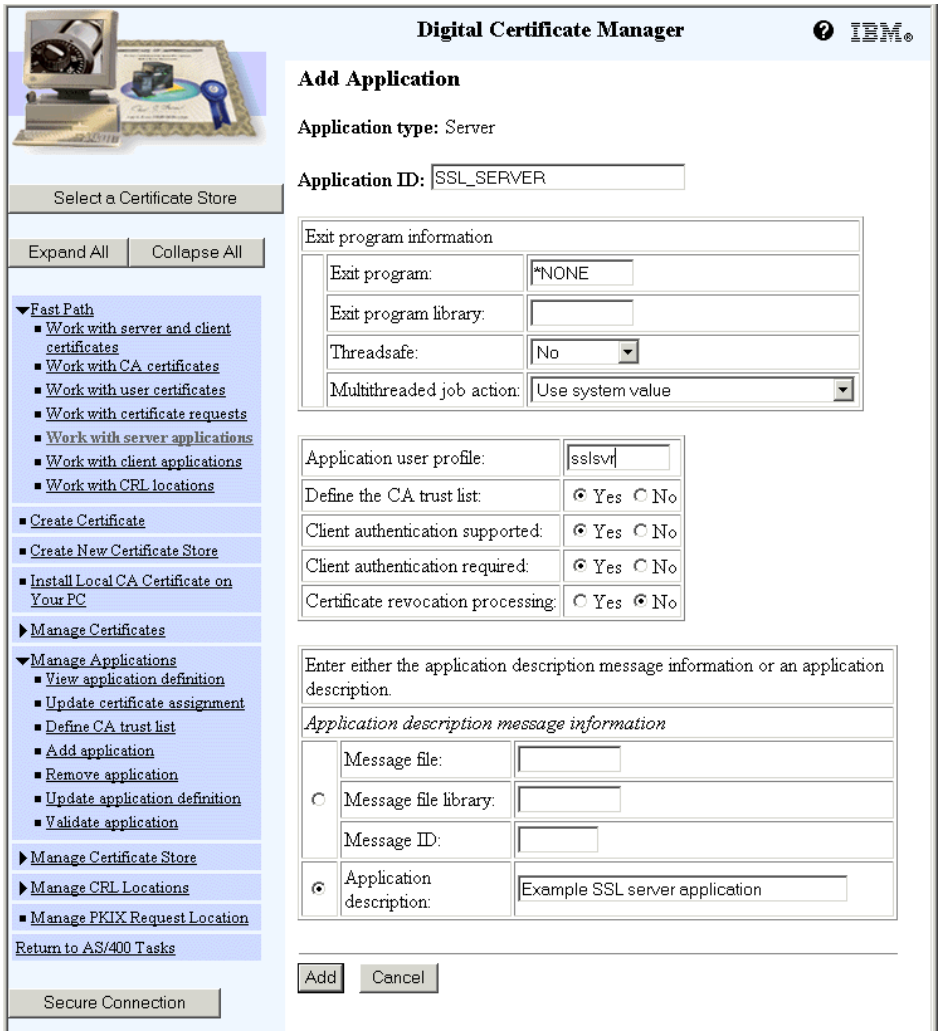


Figure 281. Entering new application details

8. Enter your application details as we have done in Figure 281.

   Application ID The application identifier is what your SSL program will use to identify the application it is using. It is required and must consist of letters, numbers, underscores, or periods. The first character must be a letter. All lowercase letters will be changed to uppercase.

   Exit program information You probably want to specify an exit program. The program named here can be used to protect your

application. It is called when requests to change your application definition in the DCM, update the CA trust list, or change the certificate assignment are performed. The program can then notify, for example, a security administrator that somebody changed the application definitions. The exit program is also called when an attempt is made to de-register the application. In this case the exit program can refuse or accept the de-register request. The exit program is not called as part of certificate processing. We have not provided examples of these exit programs.

For the exit program specifications, read *Register Application for Certificate Use* found in the iSeries Information Center by clicking **Programming->CL and APIs->APIs->Exit programs->Register Application for Certificate Use**.

Note

Another possible use of the exit program is for an application that must run continuously. Eventually the certificate assigned will expire and must be replaced. The exit program can detect the change and signal that SSL sessions must be reinitialized.

User profile
Name the user profile under which your SSL server application will run. This gives the user profile access to the *SYSTEM certificate store. If multiple user profiles will run the application, they can be given authority using Operations Navigator. This is described later under 6.5.2, “Adding a client SSL application” on page 357. It is not a good idea to try to directly grant users authority to the standard certificate stores.

Define the CA trust list
Select **Yes** if your application is going to authenticate clients based on the certificates they send and you want to specify the CAs whose certificates you will trust. **Client authentication supported** must be set to **Yes** if you set **Define CA trust list** to **Yes**.

Client authentication supported
Set this to **Yes** if you may want client applications to authenticate themselves by sending their own certificate to your server.

Client authentication required

Set this to **Yes** if you require client applications to send certificates to your server. If you select **Yes**, **Client authentication supported** must also be set to **Yes**.

> **Note**
>
> At the time of writing, if you set **Client authentication required** to **Yes** and leave **Client authentication supported** set to **No** you will get this cryptic message.
>
> ```
> Error code: 4020
> An error occurred while processing applications. Use Work with
> Problems (WRKPRB) to report the problem. Additional error
> information may be found in the HTTP server job log.
> ```
>
> Looking hard in the HTTP server joblog you find:
>
> ```
> Key 11 not allowed with value specified for key 10.
> ```
>
> This will be fixed in a future release.

Even if you set the DCM attributes to state client authentication is required, you can set the SSL attributes via the APIs to no authentication, or to optional authentication. The reverse is not true. If possible, you should set the DCM attributes to match what your application will use. A server application written to be SSL enabled with the GSK APIs needs to be aware of the following situation. If coded to use an application ID via GSK_OS400_APPLICATION_ID and GSK_SESSION_TYPE was set to GSK_SERVER_SESSION prior to the *gsk_environment_init()* call, it is possible that DCM settings will override the GSK_SESSION_TYPE. If DCM is configured with Client Authentication Required set to *Yes*, the GSK_SESSION_TYPE will be changed to be GSK_SERVER_SESSION_WITH_CL_AUTH and GSK_CLIENT_AUTH_TYPE will be set to GSK_OS400_CLIENT_AUTH_REQUIRED. The programmer can determine if this happened by calling *gsk_attribute_get_enum()* for the two enums just mentioned. It is up to the application programmer to determine what this means to their application. Note that a GSK_CLIENT_AUTH_TYPE of GSK_CLIENT_AUTH_FULL or GSK_CLIENT_AUTH_PASSTHRU will not be overridden by the DCM setting.

Table 11 shows whether client authentication will be performed based on the DCM and application settings.

Table 11. SSL and GSK API settings for client authentication

SSL/GSKit attribute for session/type client authentication	DCM Client Authentication Required *YES	DCM Client Authentication Required *NO
GSK_SERVER_SESSION	Client auth. required	No client authentication
GSK_SERVER_SESSION_WITH_CL_AUTH		
- GSK_CLIENT_AUTH_FULL	Client auth. optional	Client auth. optional
- GSK_CLIENT_AUTH_PASSTHRU	Client auth. optional	Client auth. optional
- GSK_OS400_CLIENT_AUTH_REQUIRED	Client auth. required	Client auth. required
SSL_HANDSHAKE_AS_SERVER	Client auth. required	No client authentication
SSL_HANDSHAKE_AS_SERVER_WITH_CLIENT_AUTH	Client auth. required	Client auth. required
SSL_HANDSHAKE_AS_SERVER_WITH_OPTIONAL_ CLIENT_AUTH	Client auth. optional	Client auth. optional

Certificate revocation processing

Set this to **Yes** if you want to perform CRL checking on certificates client applications send. See 2.5.1, “Managing CRL locations” on page 84, and 2.5.2, “Assigning CRL locations to CA certificates” on page 87, for more information on CRLs.

Application description

Either type a description or specify a message in a message file that contains the description.

9. Click **Add**. A message in a green box should appear stating `The application has been added` and the list of server applications should appear, including your new application, as in Figure 280.
10. Select your application and click **Work with application** at the bottom of the window. Your application details will appear in the Work with server application window.

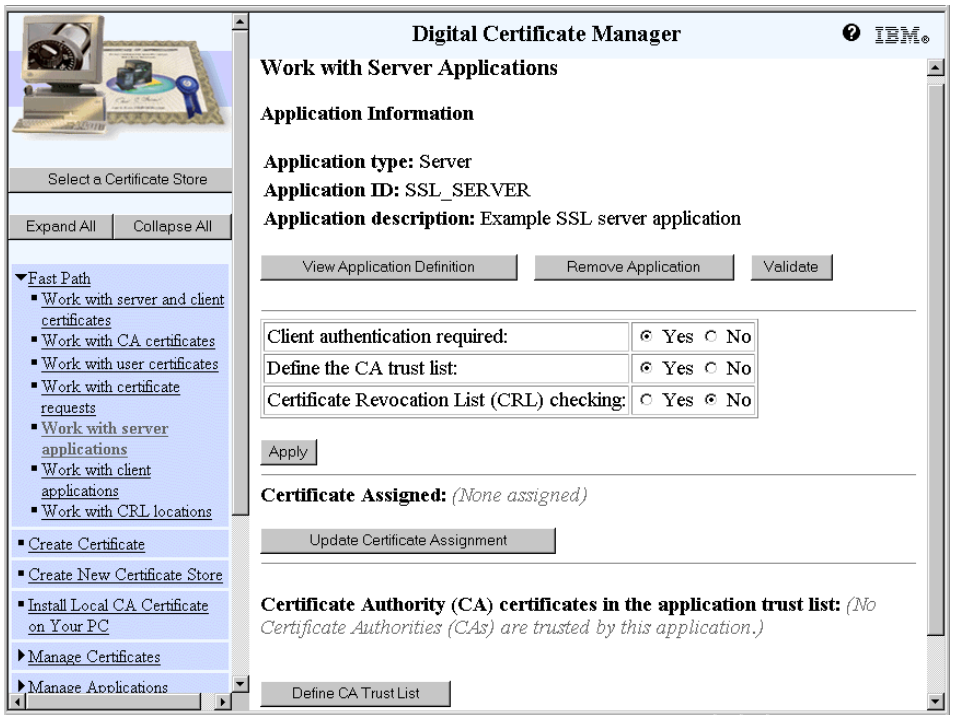


Figure 282. Work with Server Application window

11. Click **Update Certificate Assignment**. A list of all the server certificates available appears.

12. Select the certificate you want your server to present to clients. Click **Assign New Certificate**. A message, in green, appears stating `The certificate was assigned to the application`.

13. Click **Cancel**. Your application details are displayed again, as shown in Figure 282, but with the name of the certificate you assigned shown instead of *`(None assigned)`*.

14. If you want to specify the CAs you trust to sign your client's certificates, click **Define CA Trust List**. A list of all the CA certificates available appears.

    Select the certificate, or certificates, of organizations that you trust. Click **OK**. A message, in green, appears stating `Certificate Authority (CA) changes applied`.

    Click **Cancel**. Your application details are displayed again, as shown in Figure 283, but with the list of trusted CAs shown instead of *`(No Certificate Authorities (CAs) are trusted by this application)`*.

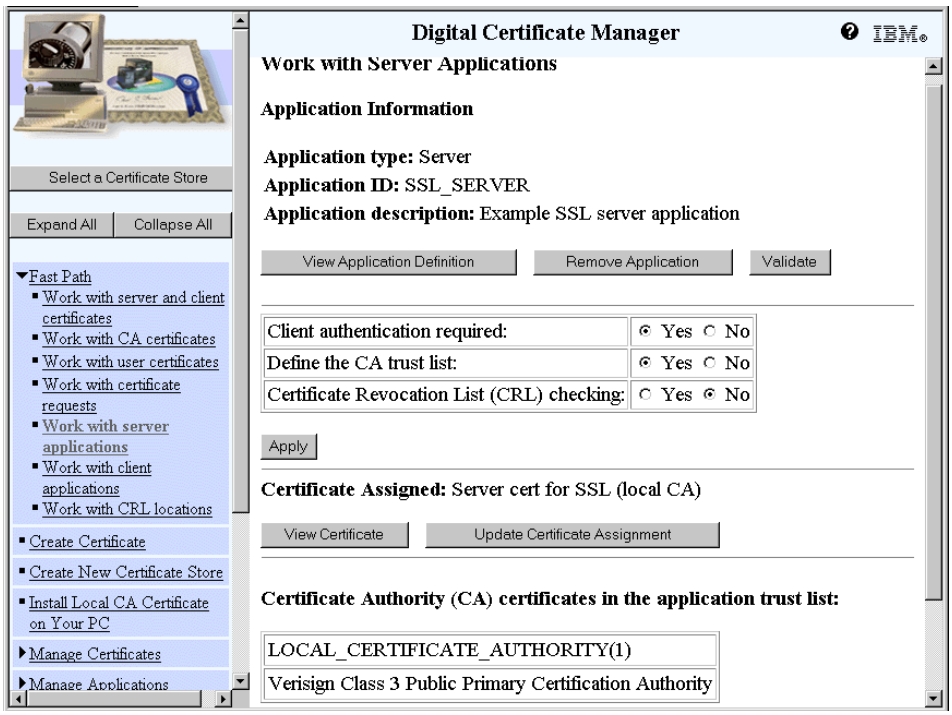


Figure 283. Work with Server Applications window

Your registration for the server application is now complete.

6.5.2 Adding a client SSL application

Adding a client SSL application is almost the same as adding a server application except for the following steps.

1. From the Fast Path options, click **Work with client applications** instead of **Work with server applications.**
2. The options relating to client authentication are not present.

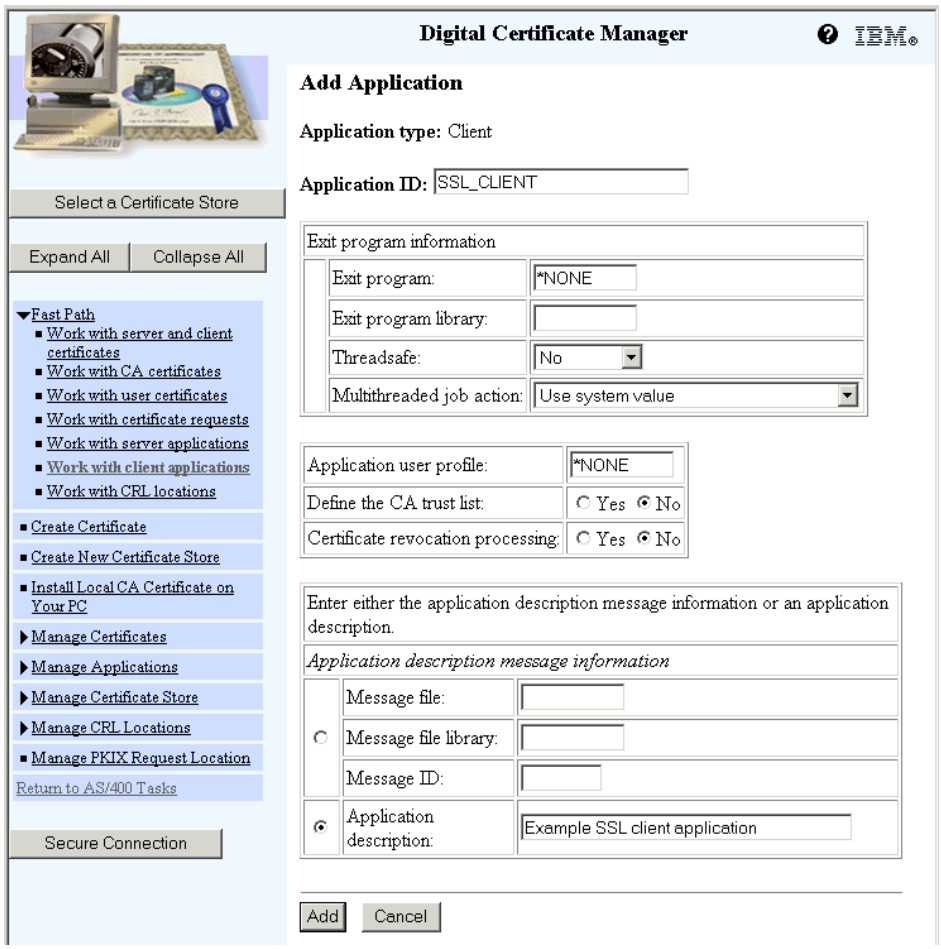


Figure 284. Adding a client application

3. You can leave the user profile as *NONE unless you are assigning a certificate to the application. In this case you should name the user profile that will be using the application so that the application can access the *SYSTEM certificate store for SSL processing.

Giving access to the *System certificate store

If multiple user profiles will use the application there are several approaches you could take to give access to the *SYSTEM certificate store:

- Create a group profile, assign it to the profiles that will use the application, and name the group profile in the DCM application definition.
- Create a single server job to do the communications processing and have pass requests to this job. There is then only one profile that needs access and this can be named in the DCM application definition.
- Have the program that uses the application adopt its owner's authority, and name its owner in the DCM application definition.
- Give the user profiles that will use the application access to the *SYSTEM certificate store for SSL processing. See Appendix B, "Granting access to the *SYSTEM certificate store" on page 411 for details of how to do this.

When a user profile is named in a Digital Certificate Manager application definition the DCM grants that profile access to the *SYSTEM certificate store for SSL processing. When the user profile is taken off the DCM application definition, or the application definition is removed, the user's access is *not* revoked. If it should be revoked, you must do this manually.

6.6 The GSK APIs

This section lists the GSK APIs grouped by function. For details of the APIs see *Secure Sockets Layer (SSL) APIs* found in iSeries Information Center by clicking **Programming->CL and APIs->APIs by category->UNIX type**. To see how to program in ILE RPG with them see the sample programs SSLSVR and SSLCLNT and the sockets prototypes in PROTO_SOC. Instructions for obtaining these are in Appendix G, "Using the additional material" on page 471.

Setting up the SSL environment

These APIs are used to set up the SSL environment that defines session characteristics used to create one or more SSL sessions later.

gsk_environment_open	Creates the environment.
gsk_attribute_set_buffer	Sets character type SSL attributes.
gsk_attribute_set_enum	Sets attributes with defined values.
gsk_attribute_set_numeric_value	Sets attributes that can take a range of numeric values.
gsk_environment_init	Initializes the environment with the attributes given.

Setting up an SSL session

These APIs are used to set up SSL sessions. An SSL session is a connection between client and server used for SSL communications. When the session is first opened it has the attributes taken from the SSL environment. Some of these can be changed before the SSL handshake is done.

gsk_secure_soc_open	Create an SSL session using the environment attributes.
gsk_attribute_set_buffer	Sets character type SSL attributes.
gsk_attribute_set_enum	Sets attributes with defined values.
gsk_attribute_set_numeric_value	Sets attributes that can take a range of numeric values.
gsk_secure_soc_init	Performs the SSL handshake using the session attributes.
gsk_secure_soc_misc	Performs miscellaneous functions for a session. May be used to redo the SSL handshake.

Obtaining SSL environment or session attributes

These APIs are used to determine SSL attributes such as the cipher that is being used or the other party's certificate.

gsk_attribute_get_buffer	Gets character type SSL attributes.
gsk_attribute_get_enum	Gets attributes with defined values.
gsk_attribute_get_numeric_value	Gets attributes that can take a range of numeric values.
gsk_attribute_get_cert_info	Gets your certificate or the other party's certificate.

Communicating between client and server

These APIs are used to read and write secured data. The normal sockets APIs can also be used to send unsecured data if this is appropriate.

However, be very careful to match the sending and receiving types - mismatched types can hang your connection.

gsk_secure_soc_write	Send secured data.
gsk_secure_soc_read	Read secured data.
send or write	Send unsecured data; send is preferred.
recv or read	Receive unsecured data.
gsk_secure_soc_startRecv	Initiate an asynchronous receive for an SSL session. This is used when handling multiple sessions concurrently. See the sections on asynchronous I/O in *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming**.
gsk_secure_soc_startSend	Initiate an asynchronous send for an SSL session. This is used when handling multiple sessions concurrently. See the sections on asynchronous I/O in *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming**.

Closing SSL down

When an SSL session will no longer be used, it should be closed to release the storage it uses. Similarly the SSL environment should be closed when no more SSL processing will be done.

gsk_secure_soc_close	Close an SSL session.
gsk_environment_close	Close the SSL environment.

Other SSL APIs

Any GSK APIs that do not fit in the preceding categories.

gsk_strerror	Get the description of a GSK return code.

6.7 SSL error handling

The GSK SSL APIs all return a result code, as most C applications do. If the API call succeeds, the return code value is @GSK_OK, which is 0. If not it will be some other value. You can see the other possible values in the API documentation or you can use the API GSK_STRERROR to obtain a pointer

to a description of the error code. It is probably a good idea to code this into your programs for debugging convenience.

Sometimes the GSK error message directs you to look at the C global error code, errno. You can use the API *errno* to do this. Again, it is best to code this into your program for debugging convenience. See the example SSL server and client programs for sample code.

Some of the errors we encountered were not immediately obvious. Following are the situations and the symptoms we saw:

- **No certificate assigned to the server application**

  Client sees GSK return code 415, `Peer not recognized or badly formatted message received`. Global C error number is 0.

  Server sees GSK return code 403, `No certificate is available for SSL processing`. Global C error number is 0.

- **No certificate assigned to the client application**

  Client sees GSK return code 202, `Key database file was not found`. C error number can have the value 3021.

  If GSK return code is not 406, then C error number should be ignored.

  Server sees GSK return code 404, `Certificate does not have a valid format`. Global C error number is 0.

- **Client does not trust server certificate or vice versa**

  Both client and server see GSK return code 6000, `Certificate is not signed by a trusted certificate authority`. Global C error number is 0.

  In the case of the client certificate being untrusted, you have the option of setting the server to use client authentication type *Pass through.* If you use this, an SSL session will be established if validation is successful or if validation fails because the certificate is self-signed, expired, or does not have a trusted root. For other validation failures, a secure session is not started. A secure session will also be started if the client does not provide a certificate. You can use the API *gsk_attribute_get_numeric_value* to detect these situations.

- **CRL checking on the certificate assigned to the client fails**

  Client sees GSK return code 406, `Error occurred in SSL processing, check the errno value`. Global C error number is 3474, `Unknown system state`.

  Server sees nothing; the client does not get to call.

6.8 Sample applications

We have used ILE RPG to create sample programs to demonstrate server and client without SSL, server and client using SSL, object signing, verifying object signatures, and retrieving object signature information. Example programs in C for sockets and SSL can be found in *Sockets programming* found in the iSeries Information Center by clicking **Programming->Programming support->Sockets programming**.

We have also created ILE RPG files defining the constants, data structures, and prototypes for the object signing APIs, many of the sockets APIs, all the SSL APIs, and a few other APIs we used along the way. This was a considerable task. If you will be programming in RPG using these APIs, we recommend you download and use these files. Instructions for obtaining them are in Appendix G, “Using the additional material” on page 471.

6.8.1 Server and client using sockets

This is a pair of ILE RPG programs that use the sockets APIs to communicate. You can run them on the same system or on two systems that have a TCP/IP connection.

Sockets server

The program *SOCSVR* is the server. This requires two parameters, the number of times it should cycle, and the return code it gives. Mostly *SOCSVR* has been written as it would to be called from another program. For testing, however, it is easiest to call it interactively and use debug to watch what it is doing.

If the program ends, because it has reached the maximum number of cycles or because it detected an error, it should close down the sockets it opened and you should be able to call it again successfully. However, if you cancel it, or if the program crashes, the sockets will be left open, which will cause errors in all subsequent calls. In this case, end the job to close the sockets before calling *SOCSVR* again.

If all works as planned, *SOCSVR* sets up a socket, waits for a client to call, reads data from the client, writes it back, closes the call socket, and waits for the next client. After the maximum number of cycles it closes the socket on which it listens, then ends.

To call *SOCSVR* to process one client call, use the command:

```
call socsvr (x'001f' x)
```

APIs used:

socket	Create a socket descriptor
setsockopt	Set socket options
bind	Associate socket with address
listen	Set socket to listen for connections
accept	Wait for connection
recv	Receive data from a socket
send	Send data to a socket
getsockopt	Get current socket options
close	Close a socket descriptor
errno	Return the C global error number

If you want to understand this program in detail, please look at the program source.

Sockets client

The program *SOCCLNT* is the client. This requires three parameters: the host name of the system where the server program is running, some text to send, and the return code it gives. Mostly *SOCCLNT* has been written to be called from another program. For testing it is easiest to call *SOCCLNT* interactively and use debug to watch what it is doing. To make interactive testing easier, the host name parameter is limited to 32 characters and the text to send parameter to 24. In production the host name parameter would probably be 255 characters or variable length and the data to send much longer.

A DNS server to return an IP address given the host name is required. If you do not have this, modify the program so you can give the IP address directly.

If all works as planned, *SOCCLNT* sets up a socket, looks up the server, connects, sends 250 bytes of text (your text 10 times) to the server, reads what the server sends back, closes the socket and ends.

To call *SOCCLNT* use the command:

```
call socclnt (server_host_name 'The text to send 2 servr' x)
```

APIs used:

socket	Create a socket descriptor
gethostbyname	Get IP address from host name
connect	Connect to sockets server

send	Send data to a socket
getsockopt	Get current socket options
recv	Receive data from a socket
close	Close a socket descriptor
errno	Return the C global error number

If you want to understand this program in detail, please look at the program source.

6.8.2 Server and client using SSL

This is a pair of ILE RPG programs that use the Global Secure Toolkit (GSK), APIs to communicate. You can run them on the same system or on two systems that have a TCP/IP connection.

SSL Sockets server

The program *SSLSVR* is the server. This requires three parameters: the number of times it should cycle, the type of client authentication to use, and the return code it gives. Mostly *SSLSVR* has been written to be called from another program. For testing however, it is easiest to call it interactively and use debug to watch what it is doing. You can also use debug to change the SSL attributes in order to try things out and to view the various SSL attributes.

If the program ends, because it has reached the maximum number of cycles or because it detected an error, it should close down the sockets it opened and you should be able to call it again successfully. However, if you cancel it or it crashes, the sockets will be left open, which will cause errors in all subsequent calls. In this case, end the job to close the sockets before calling *SSLSVR* again.

If all works as planned, *SSLSVR* sets up a socket, sets up the SSL environment, waits for a client to call, creates an SSL session, associates the call socket with the session, does the SSL handshake, checks the client certificate verification status, obtains the client's certificate, reads data from the client, writes it back, closes the SSL session, closes the call socket, and waits for the next client. After the maximum number of cycles, it closes the SSL environment and the socket on which it listens, then ends.

To call *SSLSVR* to process one client call with client authentication required, use the command:

```
call sslsvr (x'001f' r x)
```

APIs used:

- *socket* — Create a socket descriptor
- *gsk_environment_open* — Create an SSL environment
- *gsk_attribute_set_buffer* — Set SSL attributes (character)
- *gsk_attribute_set_enum* — Set SSL attributes (enumerated)
- *gsk_attribute_set_numeric_value* — Set SSL attributes (numeric)
- *gsk_environment_init* — Initialize SSL environment
- *setsockopt* — Set socket options
- *bind* — Associate socket with address
- *listen* — Set socket to listen for connections
- *accept* — Wait for connection
- *gsk_secure_soc_open* — Create an SSL session environment
- *gsk_secure_soc_init* — Start SSL handshake
- *gsk_attribute_get_buffer* — Get SSL attributes (character)
- *gsk_attribute_get_enum* — Get SSL attributes (enumerated)
- *gsk_attribute_get_numeric_value* — Get SSL attributes (numeric)
- *gsk_attribute_get_cert_info* — Get SSL certificate details
- *iconv_open* — Sets controls for CCSID conversion
- *iconv* — Do CCSID conversion
- *iconv_close* — Releases storage used for iconv
- *QsyParseCertificate* — Parse cert and return attributes
- *gsk_secure_soc_write* — Write to a secured data
- *gsk_secure_soc_read* — Read secured data
- *gsk_secure_soc_close* — Close an SSL session environment
- *close* — Close a socket descriptor
- *gsk_environment_close* — Close an SSL environment
- *errno* — Return the C global error number
- *gsk_strerror* — Return the text of a GSK error

If you want to understand this program in detail, please look at the program source.

SSL Sockets client

The program *SSLCLNT* is the client. This requires three parameters, the host name of the system where the server program is running, some text to send, and the return code it gives. Mostly *SSLCLNT* has been written to be called from another program. For testing, it is easiest to call *SSLCLNT* interactively and use debug to watch what it is doing. You can also use debug to change the SSL attributes and try things out. To make interactive testing easier the host name parameter is limited to 32 characters and the text to send parameter to 24. In production the host name parameter would probably be 255 characters or variable length and the data to send much longer.

A DNS server to return an IP address given the host name is required. If you don't have this, modify the program so you can give the IP address directly.

If all works as planned, *SSLCLNT* sets up a socket, sets up the SSL environment, looks up the server, connects, associates the call socket with the session, does the SSL handshake, sends 250 bytes of text (your text 10 times) to the server, reads what the server sends back, closes the SSL session, closes the SSL environment, closes the socket and ends.

To call *SSLCLNT* use the command:

```
call sslclnt (server_host_name 'The text to send 2 servr' x)
```

APIs used:

- *socket* — Create a socket descriptor
- *gsk_environment_open* — Create an SSL environment
- *gsk_attribute_set_buffer* — Set SSL attributes (character)
- *gsk_attribute_set_enum* — Set SSL attributes (enumerated)
- *gsk_attribute_set_numeric_value* — Set SSL attributes (numeric)
- *gsk_environment_init* — Initialize SSL environment
- *gethostbyname* — Get IP address from host name
- *connect* — Connect to sockets server
- *gsk_secure_soc_open* — Create an SSL session environment
- *gsk_secure_soc_init* — Start SSL handshake
- gsk_attribute_get_buffer — Get SSL attributes (character)
- *gsk_attribute_get_enum* — Get SSL attributes (enumerated)
- *gsk_attribute_get_numeric_value* — Get SSL attributes (numeric)
- *gsk_secure_soc_write* — Write to a secured data

- *gsk_secure_soc_read* Read secured data
- *gsk_secure_soc_close* Close an SSL session environment
- *close* Close a socket descriptor
- *gsk_environment_close* Close an SSL environment
- *errno* Return the C global error number
- *gsk_strerror* Return the text of a GSK error

If you want to understand this program in detail, please look at the program source.

6.9 SSL attributes quick lookup

This section lists all the SSL attributes that can be set or retrieved with the codes used to do so. This was complete at the time of writing. However, new attributes may be added in future releases or with PTFs. Refer to the API documentation if in doubt about the options available.

The tables are headed with the name of the API we have defined in the ILE RPG prototypes. Following each table may be notes about some of the attributes.

The table columns show:

- The names instead of the numeric value we have defined for the attributes that you can use in your program for readability.
- The number they are equivalent to.
- How the attribute can be used, S=set, G=Get, or B=both.
- A description of the attribute.

GskAtrSetBuf, GskAtrGetBuf (gsk_attribute_set_buffer, gsk_attribute_get_buffer) Set, or get, character type attributes.

Table 12. GskAtrSetBuf, GskAtrGetBuf character type attributes

Defined name	Value	Use	Description
@GSK_KEYRING_FL	201	B	Key ring file name, if not using application Id
@GSK_KEYRING_PW	202	B	Key ring password
@GSK_KEYRING_LB	203	B	Key ring file library
@GSK_V2_CIPHER	205	B	SSL V2 Cipher list
@GSK_V3_CIPHER	206	B	SSL V3 / TSL V1 Cipher list
@GSK_CONNECT_C	207	G	Cipher used in connection
@GSK_CONNECT_ST	208	G	Connection type: SSLV2, SSLV3, TSLV1
@GSK_OS400_APP	6999	B	Application ID in the DCM

The cipher lists are strings such as "2F040506" where 2F is the code for the “AES” cipher and so on. See Chapter 7, “Ciphers and cryptographic product considerations” on page 373, for more details on ciphers.

GskAtrSetEnum, GskAtrGetEnum (gsk_attribute_set_buffer, gsk_attribute_get_buffer) Set, or get, attributes that can take only defined values.

In the following table, both the attribute identifiers and the attribute values are defined. The attribute values are in italics with a “-” preceding them. For example, the attribute @GSK_PROT_SSLV2, SSLV2 supported can be set to @GSK_PROT_V2_ON or @GSK_PROT_V2_OF.

Table 13. GskAtrSetEnum, GskAtrGetEnum attributes

Defined name	Value	Use	Description
@GSK_CLIENT_AUT	401	B	Client authentication type
- *@GSK_CL_AUT_FUL*	503		Full. Certificate is not required but if given must be valid
- *@GSK_CL_AUT_PAS*	505		PassThru. Certificate is not required and session starts even if it is not validated (PGM can take action...)
- *@GSK_CL_AUT_REQ*	6995		Certificate is required and must be valid
@GSK_SESSON_TYP	402	B	Session type, client or server
- *@GSK_CL_SES*	507		Client
- *@GSK_SVR_SES*	505		Server
- *@GSK_SVR_SES_CL*	509		Server with client authentication
@GSK_PROT_SSLV2	403	B	Enable or disable SSLV2
- *@GSK_PROT_V2_ON*	510		SSLV2 supported
- *@GSK_PROT_V2_OF*	511		SSLV2 not supported
@GSK_PROT_SSLV3	404	B	Enable or disable SSLV3
- *@GSK_PROT_V3_ON*	512		SSLV3 supported
- *@GSK_PROT_V3_OF*	513		SSLV3 not supported
@GSK_PROT_TLSV1	407	B	Enable or disable TSLV1
- *@GSK_PROT_T1_ON*	518		TSLV1 supported
- *@GSK_PROT_T2_OF*	519		TSLV1 not supported
@GSK_PROT_USED	405	G	Protocol used, SSLV2, V3, or TSL
- *@GSK_PROT_V2*	514		SSL V2 is being used
- *@GSK_PROT_V3*	515		SSL V3 is being used
- *@GSK_PROT_T1*	520		TSL V1 is being used
@GSK_SID_FIRST	406	G	Cipher used in connection
- *@GSK_SID_IS_F*	516		Full SSL handshake occurred
- *@GSK_SID_NOT_F*	517		Abbreviated SSL handshake occurred

GskAtrSetNum, GskAtrGetNum (gsk_attribute_set_numeric_value, gsk_attribute_get_numeric_value) Set, or get, numeric type attributes.

Table 14. GskAtrSetNum, GskAtrGetNum attributes

Defined name	Value	Use	Description
@GSK_FD	300	B	Associated socket descriptor
@GSK_V2_TIMEOUT	301	B	SSLV2 timeout, 0-100 seconds
@GSK_V2_TIMEOUT	302	B	SSLV2 or TSLV1 timeout, 0-86400 seconds (24hrs)
@GSK_CERT_VLD	6996	G	Certificate validation code
@GSK_HS_TIMEOUT	6998	B	SSL handshake timeout, 0=forever

@GSK_CERT_VLD, the certificate validation code, is a bit out of place here. It returns defined codes such as 0 - validation successful, or 403 - GSK error, no certificate. A description of all the codes can be found either by looking up the GSK API documentation or by using the *gsk_strerror()* API. Another way of finding the description of the GSKit API error codes is by displaying the Global Secure Toolkit Layer (GSKSSL) API definition file using the command `DSPPFM FILE(QSYSINC/H) MBR(GSKSSL)`. The codes we have seen include:

0 Validation okay

403 No certificate was provided

6000 Certificate signing CA not trusted

6003 No permission to access the certificate store file

6017 The certificate has been revoked

6018 There is no CRL entry found. This error could be caused because an LDAP server was not configured/assigned or there was no CRL published for that CA on the assigned LDAP server.

Chapter 7. Ciphers and cryptographic product considerations

This chapter discusses general cryptography issues such as key length considerations and how to determine what cipher is being used. It also discusses the licensed programs you require on your iSeries or AS/400 server in order to use the Digital Certificate Manager (DCM) and to secure communications with SSL or VPN.

7.1 Licensed program products

There are several program products that provide cryptographic support. This section describes the products and how they interrelate.

7.1.1 Digital Certificate Manager, 5722SS1 Option 34

The Digital Certificate Manager (DCM) provides an interface to manage digital certificates on the iSeries or AS/400 system. This license program is required for tasks and applications that use certificates, such as HTTP servers, Telnet serves, or FTP servers. The DCM requires one of the Cryptographic Access Provider products to be installed to enable the cryptographic functions.

7.1.2 Cryptographic Access Provider, 5722-AC2 or 5722-AC3

The Cryptographic Access Provider (CAP) products are used by license programs, applications and hardware functions to enable cryptographic functions in accordance with import and export regulations. One of the Cryptographic Access Provider products needs to be installed to provide services to the DCM, the 4758 PCI Cryptographic Coprocessor for iSeries, VPN, and to applications running SSL. The 128-bit version (5722-AC3) allows you to use longer key lengths, which are much more secure than those allowed by the 56-bit version (5722-AC2). USA export restrictions formerly restricted the 128-bit version to North America.

7.1.3 Client Encryption 56-bit (5722-CE2) or 128-bit (5722-CE3)

If you want to use SSL with any Client Access Express or IBM Toolbox for Java components, you must also install one of the client encryption program products. If you install the 128-bit version you must also have installed the 128-bit version of the Cryptographic Access Provider product 5722-AC3. US export restrictions formerly restricted the 128-bit version to North America.

Note

The 40-bit Cryptographic Access Provider (5769-AC1) product and client encryption product (5769-CE1) used in releases prior to V5R1 are not supported anymore.

7.1.4 Cryptographic Service Provider, 5722-SS1 Option 35

The Common Cryptographic Architecture Cryptographic Service Provider (CCA CSP - 5722-SS1 Option 35) must be installed when using the 4758 PCI Cryptographic Coprocessor for iSeries on the iSeries server, no matter whether you use the coprocessor for speeding up the SSL handshake or write your own applications using the APIs provided with this product. In addition for the cryptographic coprocessor to work, you also need to install one of the Cryptographic Access Provider products (5722-ACx).

7.1.5 Virtual private networking

OS/400 virtual private networking (VPN) requires you to have one of the Cryptographic Access Provider products (5722-ACx) installed on the system. These products determine what encryption and authentication algorithms you can select in Internet Key Exchange (IKE) policies or data policies when configuring a VPN connection. Table 15 shows the various supported algorithms according to the Cryptographic Access Provider products.

When a VPN connection is established, the communication partners verify each other's identity. This authentication process, which involves different kinds of keying material and other variables, is performed during the IKE exchange. One of the factors that help identify the communication partner is using information that only the correct partner can know. The information can be a *preshared key* or, as introduced with V5R1, *RSA signatures* using digital certificates. To use the latter method, the Digital Certificate Manager (5722-SS1 Option 34) is required to manage the CA trust and the certificates used for authentication.

If you are doing the VPN processing on communications equipment, such as a router, then the iSeries 400 or AS/400 server encryption is not involved.

7.1.6 Cryptographic Support for AS/400, 5722-CR1

This product was initially introduced to the AS/400 server when there were no hardware cryptographic adapters available on the system. It supports data encryption and decryption, Message Authentication Code (MAC) generation

and verification, key management, and Personal Identification Number (PIN) management. It was primarily aimed to provide cryptography support that was compatible with 4700 Finance Communications Subsystems.

All functions supported by this program product can also be performed on an 4758 PCI Cryptographic Coprocessor for iSeries. However, the cryptographic coprocessor supports many more functions than the 5722-CR1 product supports.

If you do not have existing applications that use the 5722-CR1 product, you should consider using the 4758 PCI Cryptographic Coprocessor for iSeries for developing new applications.

7.1.7 Key sizes

The supported key sizes for the different protocols and program products depend on the installed Cryptographic Access Provider product. Table 15 shows the characteristics of the available Cryptographic Access Provider products on the iSeries 400 or AS/400 server.

Table 15. Supported algorithms

Algorithm	5722-AC2	5722-AC3	Notes
Hash/Authentication algorithms			
MD5	Supported	Supported	
SHA-1	Supported	Supported	
Encryption algorithms (symmetric)			
CDMF	40-bit	40-bit	2
DES	56-bit	56-bit	
3-DES	Not supported	168-bit	6
RC2	56-bit	128-bit/1024-bit	3, 7
RC4	56-bit	128-bit/2048-bit	4, 7
RC5	56-bit	128-bit	5, 7
AES	56-bit	128-bit	5, 7
SEAL	Not supported	256-bit	8
MARS	56-bit	160-bit	8
Blowfish	56-bit	448-bit	8

Algorithm	5722-AC2	5722-AC3	Notes
DESX	Not supported	64-bit	5, 7
Encryption algorithms (asymmetric)			
RSA	1024-bit	2048-bit	9
DSA	1024-bit	1024-bit	7, 9
Diffie-Hellman	1024-bit	1024-bit	7, 9

Notes:

1. The Cryptographic Access Provider product 57xx-AC1 is no longer supported in OS/400 V5R1.
2. The Common Data Masking Facility (CDMF) algorithm is supported only on the 4758 Cryptographic Coprocessor.
3. The RC2 encryption algorithm with the 5722-AC3 CAP product supports 1024-bit only when using the Java Development Kit (JDK), (5722-JV1) to access cryptography functions of the Java Cryptography Extensions (JCE). The JCE 1.2 is a standard extension to the Java 2 Software Development Kit (J2SDK), Standard Edition. The JCE implementation on iSeries 400 is compatible with the implementation of Sun Microsystems, Inc.
4. The RC4 encryption algorithm with the 5722-AC3 CAP product supports 2048-bit only when using the Java Development Kit (JDK), (5722-JV1) to access cryptography functions of the Java Cryptography Extensions (JCE). The JCE 1.2 is a standard extension to the Java 2 Software Development Kit (J2SDK), Standard Edition. The JCE implementation on iSeries 400 is compatible with the implementation of Sun Microsystems, Inc.
5. Algorithm is not supported by JCE 1.2.
6. The Triple-DES cipher is sometimes described as having a key length of 168 bits. However it is actually a 56-bit cipher performed three times, and is not as strong as some of the 128-bit ciphers such as *AES*. When we talk of 128-bit ciphers and key lengths up to 128 bits, we include Triple-DES. The 4758 Cryptographic Coprocessor supports 112-bit Triple-DES.
7. Algorithm is not supported by the 4758 PCI Cryptographic Coprocessor for iSeries.
8. Algorithm is supported *only* with the Java Cryptography Extensions (JCE) 1.2. For more information about JCE refer to *Java Cryptography Extensions* found in the iSeries Information Center by clicking **Programming->Java->IBM Developer Kit for Java->Security->Java Cryptography Extensions**.
9. The key size limitations for the public key algorithms (RSA, DSA, and Diffie/Hellman) apply to the generation of public/private key pairs. No restrictions are ever placed on the public key size when verifying a digital signature.

7.1.8 Export restrictions

Export regulations used to prohibit the export of strong encryption products from many countries. Since January 2000 when the US government relaxed the export regulations of cryptography products, most countries are now party to the *Wassenaar Arrangement* and allow the export of any encryption product to anywhere except for a few defined countries. For example, the USA forbids export to Cuba. However government rules and regulations keep changing, so in case of doubt, check first.

Due to the previous export restrictions there are still a lot of applications, primarily older Internet browsers, that can only do *weak* encryption with 56 or 40-bit keys. This may be of concern with some applications.

7.2 Upgrading to a different Cryptographic Access Provider product

If you currently have the 56-bit Cryptographic Access Provider (5722-AC2) product installed, and you install the 128-bit version (5722-AC3) to replace it, you must IPL your system before the change takes effect. Note that you can only have one 5722-ACx product installed at a time on the system, but you can install one or more client encryption (5722-CEx) products on a single system. To update a client encryption product in Client Access on a PC, you need to install the new SSL option by running the Selective Setup, with the /QIBM directory specified as the installation source directory. The setup program will remove the existing client encryption version first and then will install the new version.

Note

If the new client encryption product (for example, SSL Client Encryption 128-bit) is not displayed in the list of installable products, you need to check that the directory path in the IFS has the proper authority settings. By default, the public authority for all client encryption products is set to *EXCLUDE. If you want administrators or users to install a client encryption product on their PCs, you need to grant *RX authority to the product directory:

5722-CE2 56-bit	/qibm/proddata/CA400/Express/SSL/SSL56
5722-CE3 128-bit	/qibm/proddata/CA400/Express/SSL/SSL128

If you are also using the 4758 PCI Cryptographic Coprocessor for iSeries and want to upgrade the Cryptographic Access Provider product, you need to reload the function control vector (FCV). The FCV is a digitally signed value

stored in a file provided by IBM used by the cryptographic coprocessor to determine what type of cryptographic functions and algorithms the coprocessor supports. To update the FCV, you have to use the 4758 Cryptographic Coprocessor configuration utility from the AS/400 Tasks page or the Cryptographic_Facility_Control (CSUACFC) API when writing your own application. When using the graphical browser-based interface, select **Manage Configuration ->Attributes**, then select the desired processor and load the new FCV.

7.3 Key length considerations

This section discusses why a number of key lengths exist and how to go about choosing the most appropriate size for your intended functions.

7.3.1 Certificate keys

Certificates use public/private keys. Public/private keys are a pair of mathematically related numbers that can be used so that data encrypted with one key can only be decrypted by the other. The public key can be published so that anyone can use it without compromising the security of the system.

Part of a certificate is a public key. This can be used to encrypt data to be sent to the certificate owner, who in turn uses the corresponding private key to decrypt the data. It can also be used to decrypt data sent from the key owner, encrypted with the private key, for example to verify an object signature. The private key is, or should be, held in a secure place by the certificate owner.

A key pair is generated from the product of two large prime numbers; their security is directly related to the difficulty of determining the original primes from the product. Increasing the size of the numbers involved makes the task of determining the original primes much more difficult. A key length of 1024 bits is much harder to break than one of 512.

According to RSA Laboratories, a 512-bit key can currently be factored in a few months, and so should only be used for data that needs to be kept secure for a short period. RSA recommends 1024-bit keys for most uses and 2048-bit keys for securing extremely sensitive data such as the root key pair for a Certificate Authority. They say 768-bit keys are still secure and may be used for less valuable information.

Because the public/private key pair associated with a certificate is used for the life of the certificate, they must be very secure. If your data is valuable, it may be worthwhile for someone to invest some months of machine time to

break your keys. If you want to keep data secure for a long period, you must consider how secure your keys are likely to be at the end of that period. This is the reason for RSA recommending a key length of 1024 instead of 768.

Public/private keys have to be large enough so that it is not feasible to calculate the private key from the public key. This results in key sizes that are much larger than necessary for data encryption. If you use encryption a lot, the processing time taken may be significant. RSA Laboratories says that doubling the length of your keys will, on average, increase the time for public key operations by a factor of 4 and the time for private key operations by a factor of eight. When you renew your certificates, you should consider the key length to use. RSA Laboratories publishes its recommended key lengths on a regular basis. Refer to the URL `http://www.rsasecurity.com/rsalabs/faq/3-1-5.html` to read the article in RSA's FAQ Section 3.1.5, "How large a key should be used in the RSA cryptosystem?"

Public/private keys (asymmetric keys) are generally used for signing, verifying signatures, authentication, encrypting documents, and initiating secured communications such as SSL. They are not usually used for encrypting large amounts of data or for secured communications. The reason is the time taken for encryption and decrypting would be too long. Instead they are used to enable a secret key to be passed between the parties. The secret key (symmetric key) is then used for subsequent encryption and decryption.

7.3.2 SSL session keys

A secret key is used to secure SSL communication sessions. This is negotiated during the SSL handshake by the client. The client generates the secret key, uses the server certificate's public key to encrypt it, and sends it to the server. Then the server decrypts the encrypted secret key using its private key. Henceforth the secret key is used for encryption and decryption by both client and server.

Because the key is secret, the only issues are the security of encrypted data, and the time needed for encryption and decryption processing. Ciphers based on secret keys are chosen to give good security while requiring little time to perform encryption and decryption. They are much faster to use than public/private key pairs.

The time required to break SSL keys by one estimate are shown in Table 16 on page 380.

Table 16. Time required to break SSL keys

Key length in bits	1995	2000	2005
40	1.07 hours	8.6 minutes	68 seconds
56	7.4 weeks	6.5 days	19 hours
64	36.7 years	4.6 years	6.9 months
128	6.7e17 millennia	8.4e16 millennia	1.1e16 millenia

The numbers are very approximate, but this gives an idea of the relative security of the different key lengths. The article containing this table can be found at `http://www.tml.hut.fi/Studies/Tik-110.350/1998/Essays/ssl.html`

7.4 SSL ciphers

There are many ciphers that can be used for SSL processing. During the handshake the server and client attempt to find the best cipher they can both use. If this fails, the SSL session will not start. Both server and client may have a list of ciphers they can use. The lists are in order of most preferred to least preferred. During the SSL handshake, the client presents its list to the server and the server chooses the cipher to use - if there is at least one match.

For some SSL-enabled applications on the iSeries and AS/400 servers, you can select the ciphers you want the application to support, which includes sockets applications written with the Global Secure Toolkit (GSKit) or the HTTP Server.

In the following lists the first term in the cipher description refers to the encryption method, for example RC4. The second refers to the authentication method, for example SHA. Last is a note about where the cipher was allowed to be used and the key size. Note that the information about where a cipher can be used can change due to government regulations.

For SSL Version 3 and TLS Version 1 the ciphers supported are shown in Table 17.

Table 17. Supported ciphers in TLS V1 and SSL V3

Cipher suite	Encryption	Hash	Encryption key length
0001	Null	MD5	This performs authentication only
0002	Null	SHA	This performs authentication only
0003	RC4	MD5	Export (40-bit)
0004	RC4	MD5	US (128-bit)
0005	RC4	SHA	US (128-bit)
0006	RC2	MD5	Export (40-bit)
0009	DES	SHA	Export (56-bit)
000A	Triple-DES	SHA	US (168-bit)
002F	AES	SHA	US (128-bit)

For SSL Version 2, the ciphers supported are shown in Table 18.

Table 18. Supported ciphers in SSL V2

Cipher suite	Encryption	Encryption key length
1	RC4	US (128-bit)
2	RC4	Export (40-bit)
3	RC2	US (128-bit)
4	RC2	Export (40-bit)
6	DES	(56-bit)
7	Triple-DES	US (168-bit)

Note

The previous listing contains all the supported ciphers for the protocols specified on the iSeries and AS/400 server. The actual ciphers available for use depend on the installed Cryptographic Access Provider (CAP) product. Refer to Table 15 on page 375 for information about the ciphers supported in the different CAPs. Note that the iSeries and AS/400 server support three additional cipher suites for TLS V1 and SSL V3 (first byte contains FF) that are considered proprietary. For more information about the additional cipher suites refer to the *SSL_Init()* API documentation in the iSeries Information Center.

7.5 Controlling and determining the protocol and cipher used

Three versions of SSL, or protocols, are in use. SSL V2 and SSL V3 are standards published by Netscape. TLS V1 is an RFC (RFC2246) published and administered as are the other Internet standards.

SSL V2 is old and not often used now. It does not support client authentication and has a number of known security weaknesses. SSL V3 is significantly different from SSL V2 and is now the protocol most commonly used. TLS V1 is a new protocol, very similar to SSL V3, that addresses some security and performance issues discovered in SSL V3.

Both the SSL clients and servers have a list of ciphers, known as the *cipher suite list*, they are willing to use. During the SSL handshake, the lists are compared and a cipher, normally the strongest, is chosen.

This section discusses how you can work with protocols, cipher suite lists, and ciphers.

7.5.1 SSL applications on the iSeries 400 and AS/400 servers

Applications you create on the system can use the GSK APIs *gsk_attribute_set_buffer* and *gsk_attribute_set_enum* to specify the SSL protocols they support and the cipher suite list to be used.

If you are writing both the server and client parts of an application you may choose to use only TLS V1 and one cipher. If you are working with the public, or a number of other companies, you would probably support at least SSL V3 and TLS V1, and you would use a list of all the ciphers that are acceptable to you, in order of strongest to weakest.

SSL applications will negotiate a common protocol if they can. A server that supports TLS V1, SSL V3, and SSL V2 can communicate with clients using any of these protocols. A client supporting TLS V1 and SSL V3 can communicate with a server that support either SSL V3 or TLS V1. The only thing that does not work is a server supporting only SSL V2.

Applications can use the GSK API *gsk_attribute_get_buffer* with identifiers 207 and 208 to determine the cipher and the SSL protocol used for the current connection. To determine the cipher used from outside the SSL application would not be easy; a trace of the handshake communications and a detailed knowledge of the protocols would be required.

7.5.2 Default cipher suite lists

The default cipher list for SSL V3 and TLS V1 is 04050A090306 if the 128-bit Cryptographic Access Provider product is installed; otherwise the list is 090306.

Note

At the time this redbook was written, the V5R1 documentation stated the default list will be 2F04050A090306, that is, AES is the preferred cipher. However, IBM has decided to exclude AES from the default list until other vendors have implemented it and IBM has verified that there are no interoperability problems. When this has been done, IBM will change OS/400 to make AES the preferred cipher.

You can still use AES if you want, by specifying a cipher suite list that includes it.

For SSL V2, the default list is 136724 if the 128-bit Cryptographic Access Provider product is installed; otherwise the list is 624.

To ensure the cipher suite list your application uses, only include ciphers acceptable to you; if it is strong enough to adequately secure your data, then the cipher used is of little interest. However, you may choose to accept weak ciphers so your application can work with older browsers. In this case you may want to take some action, such as issuing a warning, if a weak cipher or protocol is being used.

7.5.3 Using SSL with Web browsers

If you are using a Web browser, you may be interested to know what cipher is being used, particularly if what you are doing is sensitive to you.

- Until recently the US government regulations prohibited export of browsers, capable of strong encryption, outside of the USA and Canada.
- Older browsers may only support weak encryption.

There are several ways of determining what cipher is used for a specific connection. One Web site we found displays the cipher you negotiate with it and can be used to determine the strongest cipher the browser supports. The URL is `http://www.fortify.net/sslcheck.html`

If you want to determine what cipher is used for a specific SSL connection, you can also display the connection properties as documented in the following examples.

7.5.3.1 Determining the used cipher with Internet Explorer 5.0

With Microsoft's Internet Explorer 5.0 you can hover the cursor over the padlock icon to display the SSL key length as shown in Figure 285.

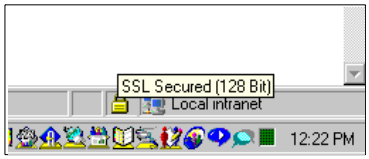


Figure 285. Microsoft Internet Explorer 5.0 - encryption key length

To obtain more details about the SSL session properties, click **File -> Properties** to display the connection details as shown in Figure 286.

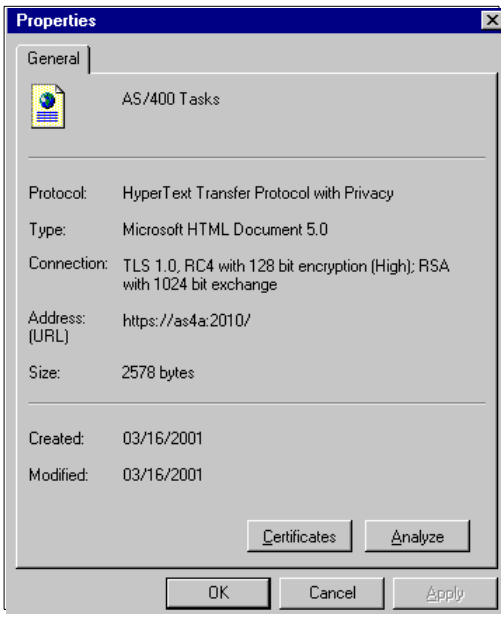


Figure 286. Microsoft Internet Explorer 5.0 SSL Properties display

The connection properties shown in Figure 286 indicate that the TLS V1 protocol with the RC4 encryption algorithm and a key length of 128 bits is used. It also states that the certificate used for authentication has a public key length of 1024 bits.

7.5.3.2 Determining the used cipher with Netscape Navigator 4.7

Perform the following steps to display the SSL connection details in Netscape Navigator V4.7:

1. Click the padlock icon or **Security** on the navigation bar of the Navigator main browser window.

Figure 287. Netscape Navigator 4.7 Security Info window 1

From this window you can do a number of tasks, including managing the certificates loaded on the browser, selecting the SSL protocols and ciphers you will use, and displaying details of the current SSL connection.

> **Note**
>
> The View Certificate button only appears when a secured connection is established.

2. To display details of the current SSL connection, click **Open Page Info**. The following window appears.

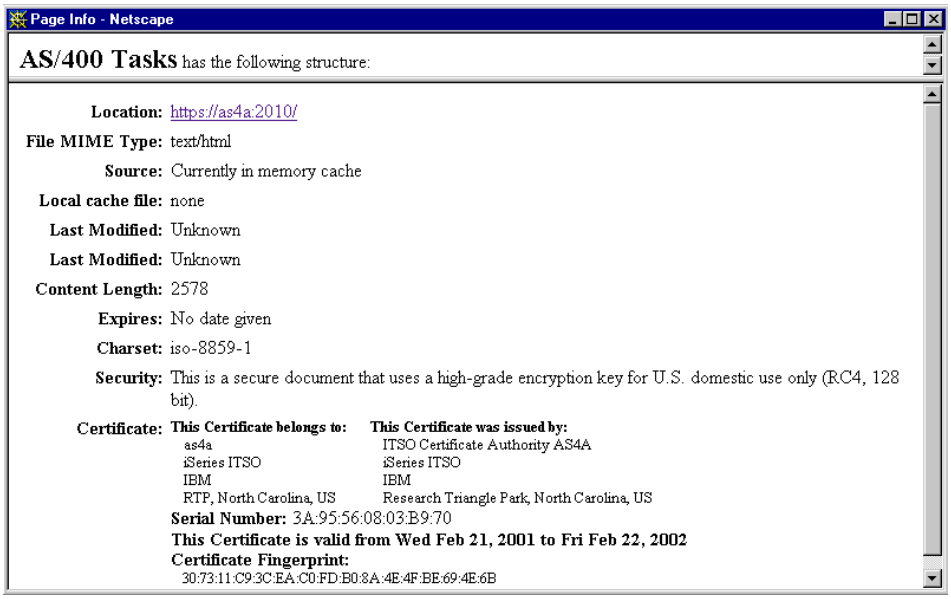


Figure 288. Netscape Navigator 4.7 SSL connection details

As shown in Figure 288 the Netscape window displays information about the used encryption algorithm as well as the key length used for encryption. Sometimes we have seen this window display “Unknown” for all the options. Internet Explorer, on the other hand, has shown SSL connection details for the same page.

7.5.3.3 Changing the security configuration in Netscape Navigator

There might be several reasons to change the default security configuration of Web browsers. This could be because you want to ensure that a certain cipher is used when connecting to a Web site or that the communication partner does not accept the proposals of the Web browser and thus you need to change the list of supported ciphers. The following steps describe how to change the security configuration in Netscape Navigator 4.7.

1. From the Navigator browser window, click the padlock icon or **Security** on the navigation bar.
2. To display and select the SSL protocols the browser can use, click **Navigator** on the security window shown in Figure 289.

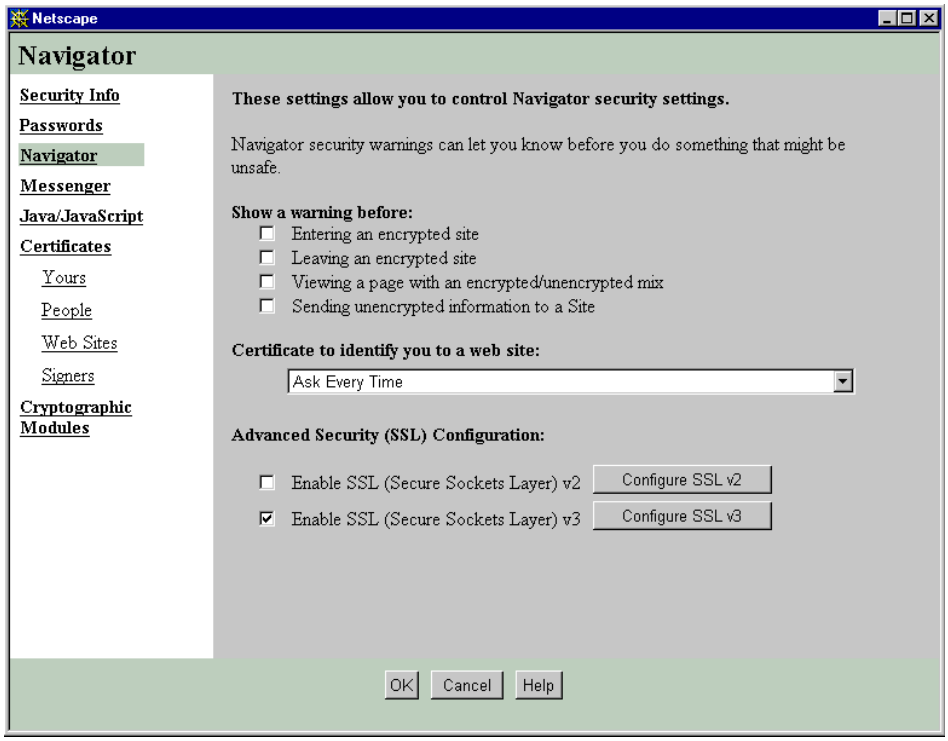


Figure 289. Display and select Navigator 4.7 SSL protocols

You can select the protocol or protocols you want the browser to support, such as SSL V2 or SSL V3. In addition you can configure the ciphers you want the browser to support as described in the next step.

3. In this example, you change the settings for SSL V3. Click **Configure SSL v3** to display the ciphers the browser can use for the protocol you have selected.

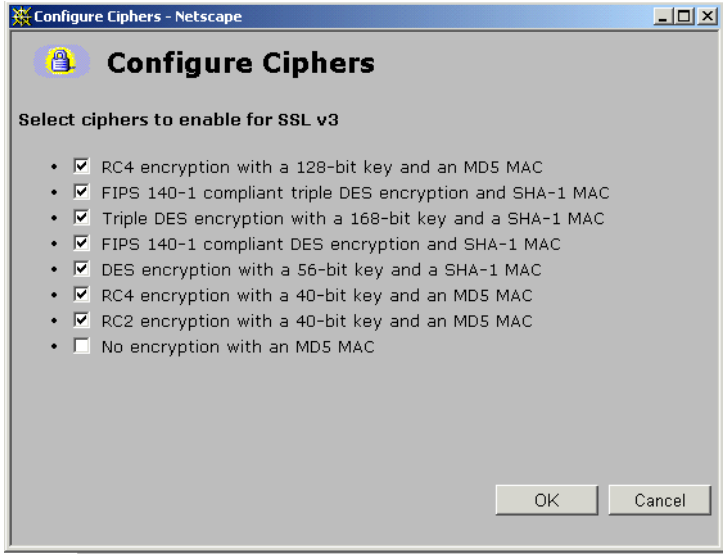


Figure 290. Netscape Navigator 4.7 SSL V3 Ciphers

You may want to unselect the weak 40-bit ciphers. However be aware that this could prevent you from communicating securely with some servers.

4. Click **OK** to save any changes you made.

7.5.3.4 Changing the security configuration in Internet Explorer

Microsoft's Internet Explorer 5.0 also has an option for selecting the protocols that can be used, which also includes TLS. However the standard configuration interface does not provide a way to select the individual ciphers you want the browser to use. Perform the following steps to change the SSL protocol settings in the Internet Explorer.

1. From the Internet Explorer browser window, click **Tools** and then **Internet Options**.
2. From the Internet Options window, click the **Advanced** tab and scroll down to **Security.**

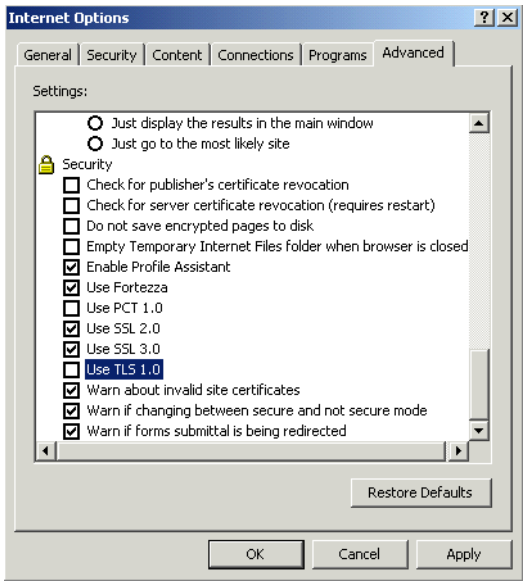


Figure 291. Select SSL protocols in Internet Explorer 5.0

3. Select the protocols, SSL 2.0, SSL 3.0, or TLS 1.0 you want the browser to be able to use and then click **OK** to save your changes.

> **Note**
>
> We experienced problems when enabling a TLS V1 connection to certain servers in the Internet. If both the server and client have correct SSL implementations, this should not happen.

As mentioned earlier in this section, the configuration interface does not provide a means of changing the list of ciphers you want the browser to use. However, we found a technical document at Microsoft's Web site that shows how to modify the registry to change the list of supported ciphers. The document was found at

`http://support.microsoft.com/support/kb/articles/Q216/4/82.ASP`

Important

Note that changing registry entries in the wrong place or setting entries to the wrong value might cause unpredictable results. Only perform the following steps when you are familiar with modifying the registry. Also consult the Microsoft Web site for recent updates to the procedure.

It is also important to understand that the changes in the list of supported ciphers affect all Windows programs that use the registry settings, not only the Internet Explorer.

The following steps summarize the necessary changes to be made to manipulate the supported cipher list.

4. Open the Windows registry with the command `regedit`.
5. Open the registry path
   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SCHANNEL\Ciphers`

   You see a list that looks the same or similar to the one shown in Figure 292.

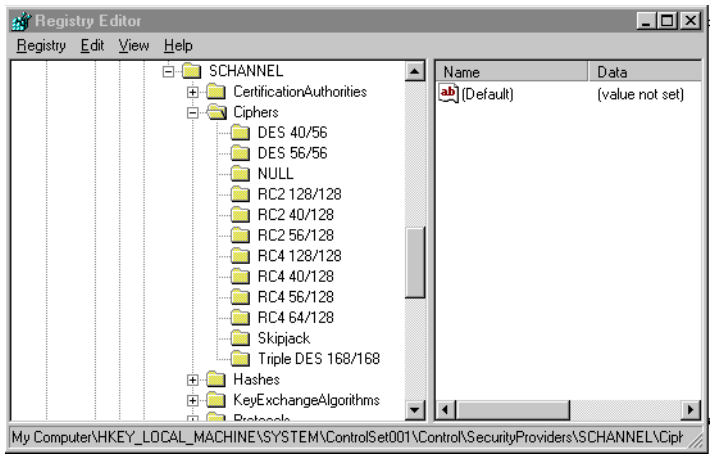


Figure 292. Cipher list in Windows registry

6. Select the cipher you want to enable or disable and double-click the **Enable** variable in the right pane of the registry window. In this case, we selected the RC4 128/128 cipher.

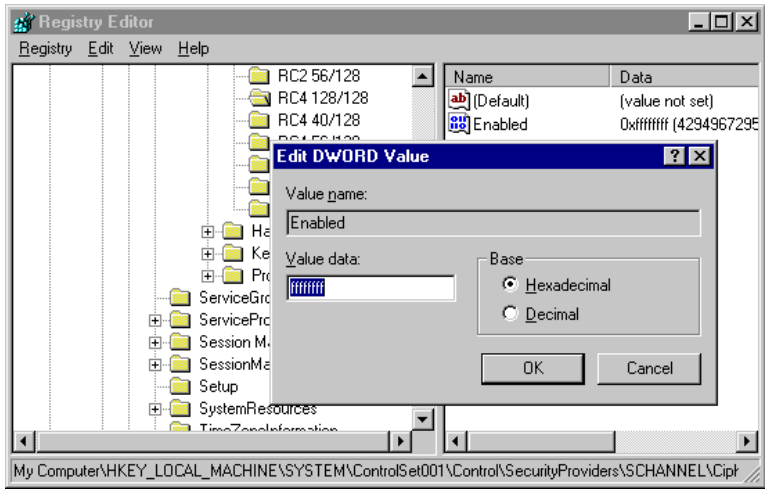


Figure 293. Updating the cipher

Before changing the value data, make sure that the value name is Enabled and the Base is set to Hexadecimal.

7. Change the value data to enable or disable the cipher according to the following values:

   0 Disabled

   ffffffff Enabled (8 times 'f')

   The example shown in Figure 293 enables the RC4 128/128 cipher for the entire system.

8. Click **OK** to save the new value and exit the registry.

7.5.4 HTTP Server (Original) and (Powered by Apache)

Both the HTTP Server (Original) and the HTTP Server (Powered by Apache) are supported on the iSeries and AS/400 server. The Apache server is an industry standard and is the recommended server for the iSeries and AS/400 servers. Both servers have directives that can be used to control the SSL protocol and cipher suite lists used.

7.5.4.1 SSL directives for the HTTP Server (Original)

There are four directives related to SSL. These are currently not configurable through the Web-based configuration and administration tool nor are they mentioned in the related help text. The directives are:

- **CipherSuiteList** <cipher-suite>
- **SSLVersion** <protocol value>
- **SSLDisableCache** <TRUE or FALSE>
- **SSLV3Timeout** <time in seconds>

CipherSuiteList <cipher-suite>
This directive specifies the list of cipher suites to be used. If this directive is not given, a default list based on the 5722-ACx product installed will be used. For more information about the hex values that can be used and their meanings, refer to 7.4, “SSL ciphers” on page 380.

For example, to support only the strongest ciphers included in the 5722-AC3 product, you need to add the following directive:

```
CipherSuiteList 2F0A0504
```

This directive defines that this server supports the AES/128-bit with SHA, 3-DES/168-bit with SHA, RC4/128-bit with SHA, and RC4/128-bit with MD5 ciphers. The preference is determined by the order of the values, from left to right.

Specify only one CipherSuiteList directive per configuration file. All supported ciphers must be added in a single string of hexadecimal characters. If you add more than one CipherSuiteList directive to the HTTP server configuration, only the last one found will be used.

OS/400 V4R3, V4R4 and V4R5 users

The CipherSuiteList directive has also been made available to OS/400 V4R3, V4R4 and V4R5 via Program Temporary Fixes (PTFs). The PTF numbers that enable this directive are:

- SF65356 V4R3
- SF62215, SF65317 V4R4
- SF62940 V4R5

The CipherSuiteList support is described in APAR SA87787.

SSLVersion <protocol value>
Specifies the protocol, or protocols, to use. The allowed values are:

ALL TLS V1 with SSL V3 and SSL V2 compatibility. TLS will be negotiated if possible, then SSL V3, then SSL V2.

SSLV2 Only SSL V2 will be allowed. (This is not recommended.)

SSLV3 Only SSL V3 will be allowed.

TLSV1 Only TLS V1 will be allowed.

TLSV1_SSLV3 TLS V1 with SSL V3 compatibility. TLS will be negotiated if possible, then SSL V3. If neither can be negotiated the handshake will fail.

If this directive is not specified or is invalid, the SSLVersion defaults to ALL. For example, to require that TLS V1 be used, specify:

```
SSLVersion TLSV1
```

SSLDisableCache <TRUE or FALSE>
This directive indicates whether SSL should allow SSL session information to be cached for use in future SSL handshakes. The default is to use caching.

If caching is allowed, abbreviated SSL handshake processing will be attempted on subsequent SSL connections from the same client after an initial full SSL handshake is successful. When a client and the server want to resume an existing session, the client sends a ClientHello message including the existing session ID to the server. If the session ID is cached on the server and no SSL time-out has occurred, the session will be resumed. Some people may want to force a full handshake every time to fully authenticate the endpoints.

For example, to disable SSL caching, specify:

```
SSLDisableCache TRUE
```

SSLV3Timeout <time in seconds>
The time, in seconds, to hold cached SSL session information. This applies to both SSL V3 and TLS V1 connections. The default is 24 hours (86400 seconds). If cached session information times out, the next connection will require a full SSL handshake.

For example, specify:

```
SSLV3Timeout 3600
```

Note that the SSLV3Timeout directive will be ignored if SSL caching is disabled through the SSLDisableCache directive.

Entering the directives
Because the new directives are not supported by the configuration and administration tool, you have to use the HTTP configuration editor by performing the OS/400 command `WRKHTTPCFG CFG(name)` to enter them. This

starts a simple editor you can use to maintain your configuration. Using this editor is not recommended since it is possible, indeed easy, to enter incorrect or out of sequence directives and create errors that are difficult to locate. Use the configuration tool when possible.

The following steps guide you through the configuration of the new SSL directives using the HTTP configuration editor:

1. At an OS/400 command prompt enter:

```
WRKHTTPCFG CFG(EXAMPLE)
```

```
                        Work with HTTP Configuration
                                                              System:   AS4A
  Configuration name . . . . . . . :   EXAMPLE

Type options, press Enter.
  1=Add   2=Change   3=Copy   4=Remove   5=Display   13=Insert

      Sequence
Opt    Number    Entry
1      115       CipherSuiteList 2F04050A
       00030     HostName HARDCODEDALL
       00040     Port 80
       00050     Pass /* /www/example/*
       00060     NormalMode On
       00070     # Do not change or delete the following AppName directi   >
       00080     AppName QIBM_HTTP_SERVER_EXAMPLE
       00090     SSLMode On
       00100     SSLPort 443
       00110     SSLClientAuth On

                                                                     Bottom
F3=Exit   F5=Refresh   F6=Print List   F12=Cancel   F17=Top   F18=Bottom
F19=Edit Sequence
```

Figure 294. Adding cipher suite list directive 1

Specify your new directive on the top line by typing 1 for the option, the appropriate sequence number, and the start of your directive. In this example it is the CipherSuiteList directive:

```
CipherSuiteList 2F04050A
```

2. Press **Enter**.

   A second display appears. On this you can enter further text for the directive. You can also use F11 to enter more directives, but we do not recommend this because you may lose track of the sequence numbers and where the new directives are placed.

```
                         Add HTTP Configuration Entry
                                                                System:   AS4A
 Sequence Number  . . . . . . . . . . . .   00115
 Entry  . . . . . . . . . . . . . . . . .   CipherSuiteList 2F04050A




                                                                          Bottom
 Press Enter to complete.

 F3=Exit   F11=Another Entry   F12=Cancel
```

Figure 295. Adding cipher suite list directive 2

Complete your directive.

3. Press Enter again to complete the process. The configuration list is redisplayed including your new directive.

7.5.4.2 SSL directives for the HTTP Server (Powered by Apache)

The SSL directives for the HTTP Server (Powered by Apache) can be set manually by editing the server configuration or by using the Web-based HTTP Server configuration and administration tool, which is the easiest and preferred way. This server basically supports the same options with some differences in the syntax. The following directives can be used to control certain attributes during an SSL handshake:

```
SSLCipherSpec <cipher-spec>

SSLVersion <protocol value>

SSLCacheEnable

SSLCacheDisable

SSLV2Timeout <time in seconds>

SSLV3Timeout<time in seconds>
```

The options for controlling SSL are grouped under the heading "Authentication and Security" in the HTTP Server configuration and administration tool.

In the following example, it is assumed that the HTTP server is already configured for SSL and a certificate assigned. For more information about enabling SSL for the HTTP Server (Powered by Apache), refer to Chapter 5, “Securing OS/400 application traffic with SSL/TLS” on page 283.

Configuring SSL caching and timeouts with the graphical interface

The steps in this section describe how to configure the SSL caching and the timeout values for SSL Version 2 and 3 using the HTTP Server configuration and administration tool.

1. Start the configuration and administration tool from the AS/400 Tasks page by selecting the links **IBM HTTP Server for AS/400** and then **Configuration and Administration**.
2. Select the **Configuration** tab and choose your existing server configuration from the pull-down list.

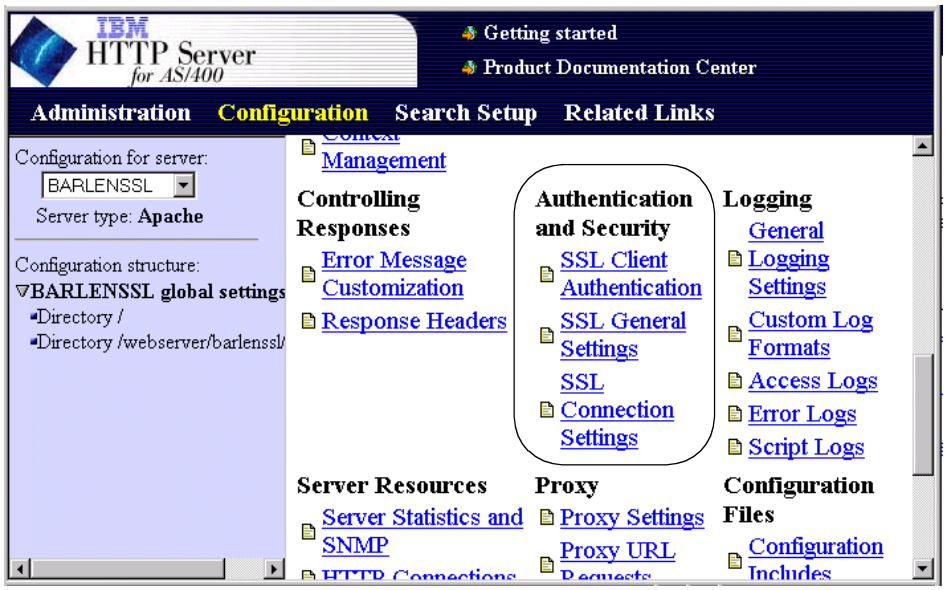


Figure 296. Configuration and Administration tool

3. Click **SSL General Settings** to enable SSL, to set the name of the DCM application to use, and to set the SSL caching attributes.

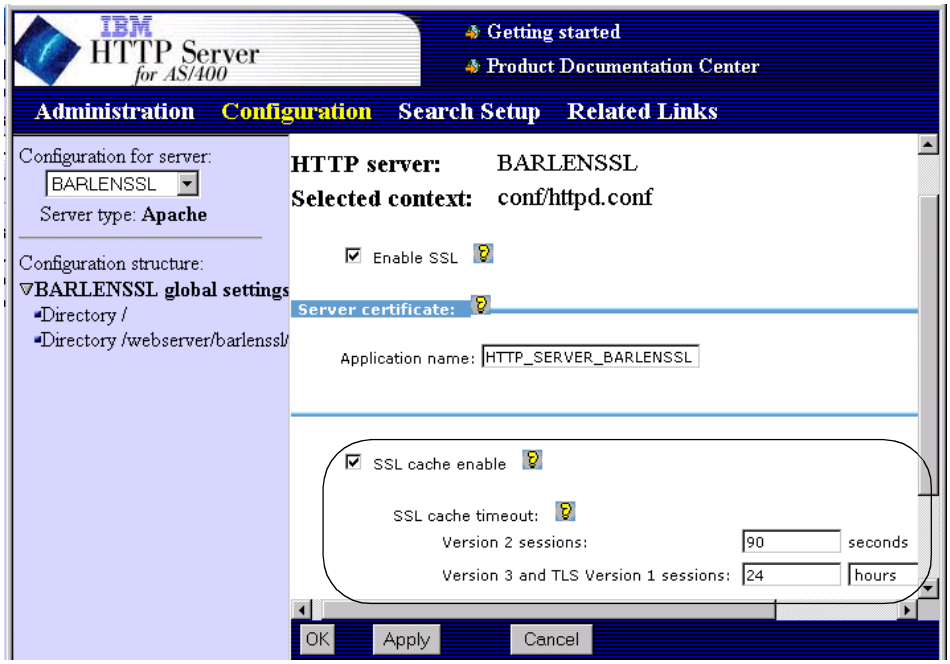


Figure 297. Apache HTTP server general SSL settings

On the bottom half of the configuration form, you can define whether SSL session caching is enabled and what SSL Version 2 and 3 timeout values will be used. The following list contains information about the possible input values and the corresponding directives as automatically added by the configuration and administration tool to the configuration file.

- `SSLCacheEnable` **and** `SSLCacheDisable`

  These directives indicates whether SSL should allow SSL session information to be cached for use in future SSL handshakes. The default when configuring SSL is to use caching.

  If caching is allowed, abbreviated SSL handshake processing will be attempted on subsequent SSL connections from the same client after an initial full SSL handshake is successful. When a client and the server want to resume an existing session, the client sends a ClientHello message including the existing session ID to the server. If the session ID is cached on the server and no SSL timeout has occurred, the session will be resumed. Some people may want to force a full handshake every time to fully authenticate the endpoints.

For example, to disable SSL caching, specify:

`SSLCacheDisable` (SSL Cache enable is unselected)

`SSLCacheEnable` (SSL Cache enable is selected)

`-SSLV2Timeout <time in seconds>`

This directive indicates the time, in seconds, to hold cached SSL session information. This applies to SSL V2 connections only. The default is 100 seconds. The allowed input range goes from 1 to 100 seconds. Entering values greater than 100 seconds in the configuration interface defaults the SSLV2Timeout back to 100 seconds. If cached session information times out, the next connection will require a full SSL handshake.

For example, specify:

```
SSLV2Timeout 60
```

Note that the SSLV2Timeout directive will be deleted from the HTTP server configuration when caching is disabled through the SSLCacheDisable directive.

`-SSLV3Timeout <time in seconds>`

This directive indicates the time, in seconds, to hold cached SSL session information. This applies to both SSL V3 and TLS V1 connections. The default is 24 hours (86400 seconds). The input range is from 1 to 86400 seconds. If cached session information times out, the next connection will require a full SSL handshake.

For example, specify:

```
SSLV3Timeout 3600
```

Note that the SSLV3Timeout directive will be deleted from the HTTP server configuration when caching is disabled through the SSLCacheDisable directive.

Note

The SSL timeout values have a direct impact on performance. The lower this value is, the more secure the connection is, but performance will suffer if the value is lower, since a full handshake is much slower than an abbreviated handshake.

4. Click **OK** to save your changes. A server restart is required to put the changes into effect.

Configuring SSL protocols and ciphers with the graphical interface

The following steps describe how to control the protocols and ciphers that this HTTP server accepts. The configuration is performed through the HTTP Server configuration and administration tool.

1. Start the configuration and administration tool from the AS/400 Tasks page by selecting the links **IBM HTTP Server for AS/400** and then **Configuration and Administration**.
2. Click the **Configuration** tab and select your existing server configuration from the pull-down list.
3. Select **SSL Connection Settings** from the global settings page to define the list of supported protocols and ciphers. A configuration form as shown in Figure 298 appears.

.

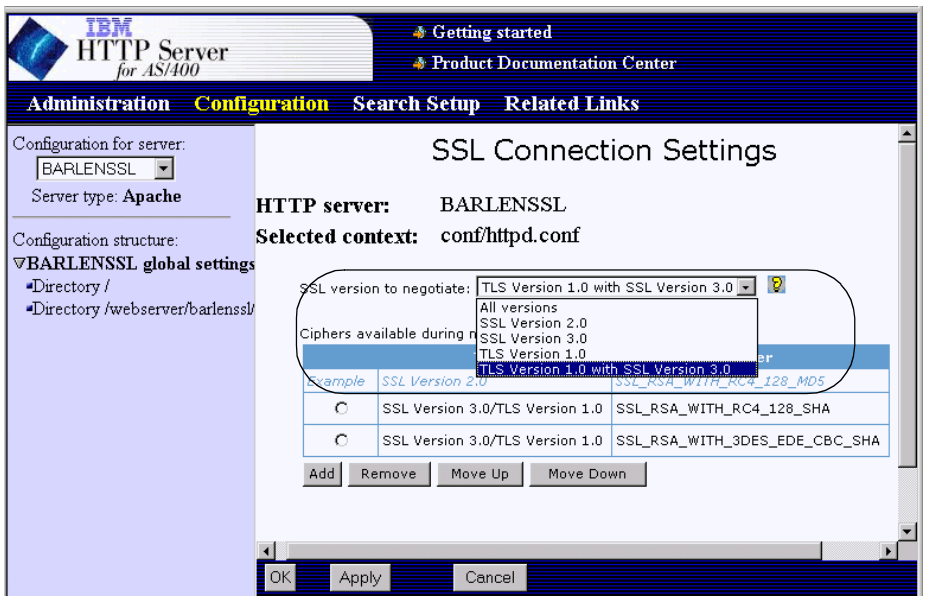


Figure 298. Apache HTTP server SSL protocols and ciphers

To control the protocol used, open the drop-down box headed **SSL version to negotiate** and select the protocol option you want. The listed values are:

- All versions
- SSL Version 2.0
- SSL Version 3.0
- TLS Version 1.0
- TLS Version 1.0 with SSL Version 3.0

These correspond to:

```
SSLVersion <protocol value>
```

This directive specifies the protocol, or protocols, to use. The allowed values are:

`ALL`	TLS V1 with SSL V3 and SSL V2 compatibility. TLS will be negotiated if possible, then SSL V3, then SSL V2.
`SSLV2`	Only SSL V2 will be allowed. (This is not recommended.)
`SSLV3`	Only SSL V3 will be allowed.
`TLSV1`	Only TLS V1 will be allowed.
`TLSV1_SSLV3`	TLS V1 with SSL V3 compatibility. TLS will be negotiated if possible, then SSL V3. If neither can be negotiated, the handshake will fail.

If this directive is not specified or is invalid, the SSLVersion defaults to ALL.

For example, to require that TLS V1 be used, specify:

```
SSLVersion TLSV1
```

To control the ciphers used, click **Add** under the section titled "Ciphers available during negotiation" to add one entry to the list of ciphers to be used. Open the drop-down boxes on the new entry and select the protocol and cipher you want. Repeat until you have a list of all the ciphers you want to use.

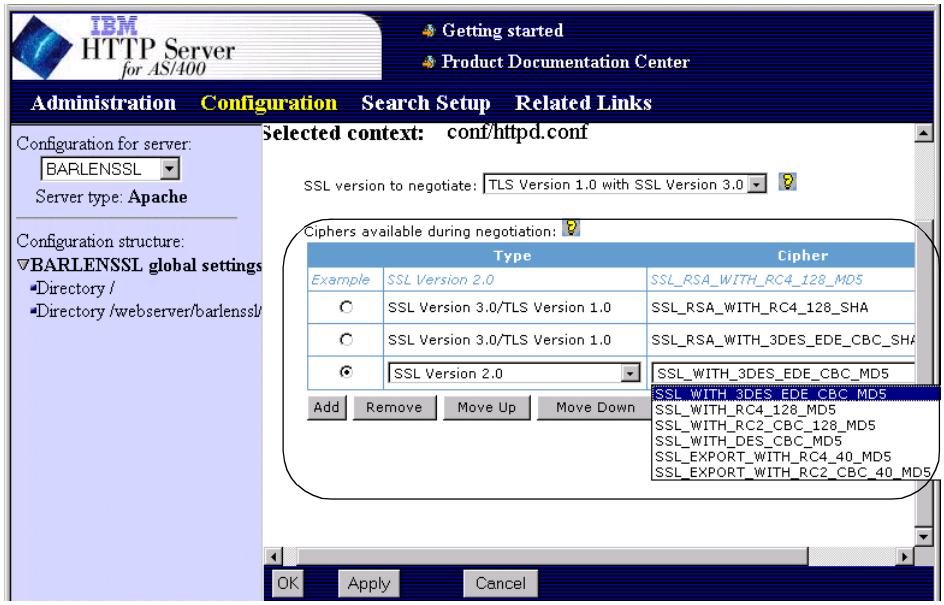


Figure 299. Apache HTTP server SSL protocols and ciphers 2

Ensure the list is ordered with the ciphers you prefer at the top. To move entries up or down the list, select them, then click **Move Up** or **Move Down**. For each of the ciphers added to the list a SSLCipherSpec directive is added to the configuration. The SSLCipherSpec syntax is as follows:

```
SSLCipherSpec <cipher-spec>
```

This directive specifies a single cipher that this HTTP server instance supports. In a typical implementation, you would support more than one cipher. This is accomplished by adding several SSLCipherSpec statements to the configuration, one per cipher spec, ordered from top to bottom from the most preferred cipher to the least. If this directive is not given, a default list based on the 5722-ACx product installed will be used. For more information about the ciphers that can be used with the individual protocols, refer to 7.4, “SSL ciphers” on page 380.

The following example shows a configuration that supports the RC4 encryption algorithm with 128-bit key length using the SHA authentication algorithm and the Triple-DES encryption algorithm with 128-bit key length also using the SHA authentication algorithm:

```
SSLCipherSpec SSL_RSA_WITH_RC4_128_SHA
SSLCipherSpec SSL_RSA_WITH_3DES_EDE_CBC_SHA
```

In this example, the RC4 algorithm is the preferred cipher. This means if a client sends a list of supported ciphers that contains both ciphers mentioned, the RC4 cipher will be selected by the HTTP server.

4. Click **OK** at the bottom of the window to save your changes. Restart the server to activate them.

7.6 Other applications that use SSL on the iSeries and AS/400 servers

Many other applications on the system can use SSL such as Telnet, FTP, or WebSphere. Most of the applications provided by IBM use the DCM to store their certificates and provide services related to certificates, and use the Cryptographic Access Provider products to enable the algorithms and their key lengths that can be used for encryption and decryption.

Applications ported to the iSeries or AS/400 servers may have their own routines for SSL and cryptography. However, they will probably still store their certificates in *key ring files*, that is user certificate stores, which the DCM can administer.

Currently the HTTP servers are the only IBM provided products for which you can control the protocol and ciphers used.

WebSphere use of DCM and 5722ACx program products

The WebSphere Application Server (WAS) is an example of an application ported to the AS/400 server with its own routines for working with digital certificates and encryption. If you use the WAS Advanced Edition, which connects directly to its clients using Enterprise JavaBeans, then authentication and encryption is done by WAS. However, if you use the HTTP server to access applications running under WAS, SSL authentication and encryption is done by the HTTP sever in the normal way. Also, when the WAS administration tool is used to publish information to an LDAP directory, the DCM and Cryptographic Access Provider program products are used.

Appendix A. 4758 cryptographic coprocessor hardware commands

The 4758 PCI Cryptographic Coprocessor for iSeries has access controls in place that allow a security administrator to grant access to security tasks to individual profiles. This is accomplished by adding security tasks to a group called *role*. Later the administrator can assign roles to profiles. The information in Table 19 provides an overview of all available hardware commands and their association to the standard roles as they are created when using the 4758 Cryptographic Coprocessor configuration wizard and you have selected that three profiles are to be created. If you have chosen to operate the 4758 coprocessor with one profile, only the Default and CRYPMSTR roles are created. In the latter case, the CRYPMSTR has the authority to perform all hardware commands.

Table 19. 4758 Cryptographic Coprocessor commands and roles

Command description	**Cmd code**	**Standard roles when three profiles are created**			
		CRYPMSTR	**CRYPSEC**	**CRYPADMN**	**Default**
Encipher	000E	Y	Y	Y	Y
Decipher	000F	Y	Y	Y	Y
Generate MAC	0010	Y	Y	Y	Y
Verify MAC	0011	Y	Y	Y	Y
Re-encipher to Master Key	0012	Y	Y	Y	Y
Re-encipher from Master Key	0013	Y	Y	Y	Y
Load 1st Master Key Part (symmetric)	0018	Y			
Combine Master Key Parts - middle/last (symmetric)	0019	Y			
Set Master Key (symmetric)	001A	Y			
Load First Key Part	001B	Y	Y	Y	Y
Combine Key Parts	001C	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
Compute Verification Pattern (N/A AS/400)	001D	Y			
Translate Key	001F	Y	Y	Y	Y
Generate Random Master Key	0020	Y			
Clear New Master Key (symmetric)	0032	Y			
Clear Old Master Key (symmetric)	0033	Y			
Generate Diversified Key	0040	Y	Y	Y	Y
Load 1st Master Key Part (asymmetric)	0053	Y			
Combine Master Key Parts - middle/last (asymmetric)	0054	Y			
Set Master Key (asymmetric)	0057	Y			
Clear New Master Key (asymmetric)	0060	Y			
Clear Old Master Key (asymmetric)	0061	Y			
Generate Key Set	008C	Y	Y	Y	Y
Generate Key	008E	Y	Y	Y	Y
Re-encipher to Current Master Key	0090	Y	Y	Y	Y
Generate Clear 3624 PIN	00A0	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
Generate Clear 3624 PIN Offset	00A4	Y	Y	Y	Y
Verify Encrypted 3624 PIN	00AB	Y	Y	Y	Y
Verify Encrypted GPB PIN	00AC	Y	Y	Y	Y
Verify Encrypted VISA PVV	00AD	Y	Y	Y	Y
Verify Encrypted InterBank PIN	00AE	Y	Y	Y	Y
Format and Encrypt PIN	00AF	Y	Y	Y	Y
Generate Formatted and Encrypted GBP PIN	00B1	Y	Y	Y	Y
Generate Formatted and Encrypted Inter-Bank PIN	00B2	Y	Y	Y	Y
Translate PIN with No Format Control to No Format Control	00B3	Y	Y	Y	Y
Reformat PIN with No Format Control to No Format Control	00B7	Y	Y	Y	Y
Generate Clear VISA PVV Alternate	00BB	Y	Y	Y	Y
Encipher Under Mas-ter Key	00C3	Y	Y	Y	Y
Encipher Under Mas-ter Key Extended	00C4	Y	Y	Y	Y
Lower Export Authority	00CD	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
Translate Control Vector	00D6	Y	Y	Y	Y
Generate Key Set Extended	00D7	Y	Y	Y	Y
Encipher Crypto Variable	00DA	Y	Y	Y	Y
Replicate Key	00DB	Y	Y	Y	Y
Generate CVV	00DF	Y	Y	Y	Y
Verify CVV	00E0	Y	Y	Y	Y
Derive Key For Unique Key Per Transaction	00E1	Y	Y	Y	Y
Generate Initial PEK for UKPT PIN Pad	00E2	Y	Y	Y	Y
Digital Signature Generate	0100	Y			Y
Digital Signature Verify	0101	Y	Y	Y	Y
Key Token Change	0102	Y			Y
PKA Key Generate	0103	Y			Y
PKA Key Import	0104	Y	Y	Y	Y
Symmetric Key Export	0105	Y	Y	Y	Y
Symmetric Key Import	0106	Y	Y	Y	Y
One Way Hash	0107	Y	Y	Y	Y
Data Key Import	0109	Y	Y	Y	Y
Data Key Export	010A	Y	Y	Y	Y
Compose SET Block	010B	Y	Y	Y	Y
Decompose SET Block	010C	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
PKA92 Symmetric Key Generate	010D	Y	Y	Y	Y
NL-EPP-5 Symmetric Key Generate	010E	Y	Y	Y	Y
Reset Intrusion Latch	010F			Y	
Set Clock	0110			Y	
Reinitialize Device	0111			Y	
Initialize Access Control	0112		Y		
Change Expiration Date Of Profile	0113		Y		
Change Passphrase	0114		Y		
Reset Logon Failure Count	0115		Y		
Load Roles And Profiles	0116		Y		
Delete Profile	0117		Y		
Delete Role	0118		Y		
Load FCV	0119			Y	
Clear FCV	011A			Y	
Force User Logoff	011B			Y	
Set EID	011C			Y	
Initialize Master Key Cloning	011D	Y			
RSA Encipher Clear Data	011E	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
RSA Decipher Clear Data	011F	Y	Y	Y	Y
Generate Random Master Key (asymmetric)	0120	Y			
Register PKA Public Key Hash	0200	Y			
Register PKA Public Key with Cloning	0201	Y			
Register PKA Public Key	0202	Y			
Delete Retained Key	0203	Y	Y	Y	Y
PKA Clone Key Generate	0204	Y			Y
PKA Clone Key Generate Part2	0205	Y			Y
Clone Info Obtain 1	0211	Y			
Clone Info Obtain 2	0212	Y			
Clone Info Obtain 3	0213	Y			
Clone Info Obtain 4	0214	Y			
Clone Info Obtain 5	0215	Y			
Clone Info Obtain 6	0216	Y			
Clone Info Obtain 7	0217	Y			
Clone Info Obtain 8	0218	Y			
Clone Info Obtain 9	0219	Y			
Clone Info Obtain 10	021A	Y			
Clone Info Obtain 11	021B	Y			

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
Clone Info Obtain 12	021C	Y			
Clone Info Obtain 13	021D	Y			
Clone Info Obtain 14	021E	Y			
Clone Info Obtain 15	021F	Y			
Clone Info Install 1	0221	Y			
Clone Info Install 2	0222	Y			
Clone Info Install 3	0223	Y			
Clone Info Install 4	0224	Y			
Clone Info Install 5	0225	Y			
Clone Info Install 6	0226	Y			
Clone Info Install 7	0227	Y			
Clone Info Install 8	0228	Y			
Clone Info Install 9	0229	Y			
Clone Info Install 10	022A	Y			
Clone Info Install 11	022B	Y			
Clone Info Install 12	022C	Y			
Clone Info Install 13	022D	Y			
Clone Info Install 14	022E	Y			
Clone Info Install 15	022F	Y			
List Retained Key	0230	Y	Y	Y	Y
Generate Clear NL-PIN-1 Offset	0231	Y	Y	Y	Y
Verify Encrypted NL-PIN-1 Offset	0232	Y	Y	Y	Y

Command description	Cmd code	Standard roles when three profiles are created			
		CRYPMSTR	CRYPSEC	CRYPADMN	Default
PKA92 Symmetric Key Import	0235	Y	Y	Y	Y
PKA92 PIN Key Import	0236	Y	Y	Y	Y
Zero-pad Symmetric Key Generate	023C	Y	Y	Y	Y
Zero-pad Symmetric Key Import	023D	Y	Y	Y	Y
Zero-pad Symmetric Key Export	023E	Y	Y	Y	Y
PKCS-1.2 Symmetric Key Generate	023F	Y	Y	Y	Y

Appendix B. Granting access to the *SYSTEM certificate store

When you run an application that uses a certificate, you need to access the key database file holding the certificate. For SSL using applications defined in the Digital Certificate Manager (DCM), this file is the *SYSTEM certificate store.

The best way to grant a user profile access to the *SYSTEM certificate store for SSL processing is by using Operations Navigator. There is a specific option just for this. It is not recommended that you give access by granting authority to the key database file and to all the directories in the path to the file. This is because it is cumbersome, it gives users authority to several directories that they should not use, and IBM may change the DCM implementation later so it no longer works at all.

When a user profile is named on a DCM application definition, the DCM grants that profile access to the *SYSTEM certificate store for SSL processing. When the user profile is taken off the DCM application definition, or the application definition is removed, the user's access is *not* revoked. If it should be revoked, you must do this manually.

Note

There are two ways of granting access to the *SYSTEM certificate store with Operations Navigator. The first way, described in this section, shows a way to authorize individual user profiles to one or more applications or stores. The second way is to grant access for one application or certificate store to multiple users at the same time. If you want to read more about the second approach, refer to 3.3.5, "Authorizing users to use object signing applications" on page 137. For the second method, just select the *SYSTEM certificate store instead of the object signing application.

To give a user profile access to the *SYSTEM certificate store using Operations Navigator, complete these steps:

1. Start Operations Navigator.
2. Open your AS/400 system.
3. Double-click **Users and Groups** to expand.

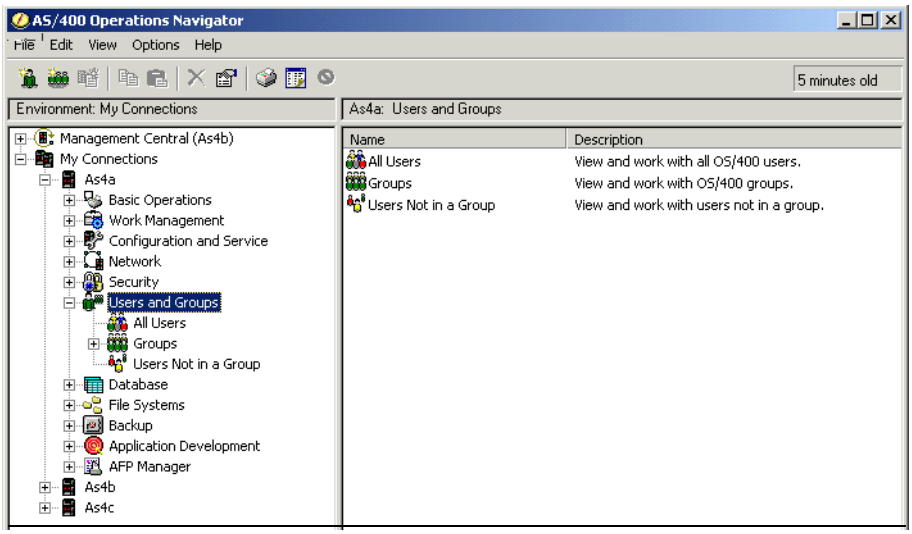


*Figure 300. Giving access to the *SYSTEM certificate store 1*

4. Double-click **All Users**, or **Groups,** to expand.
5. Double-click the user profile or group profile you want to give access to so that their properties are displayed, as shown in Figure 301.

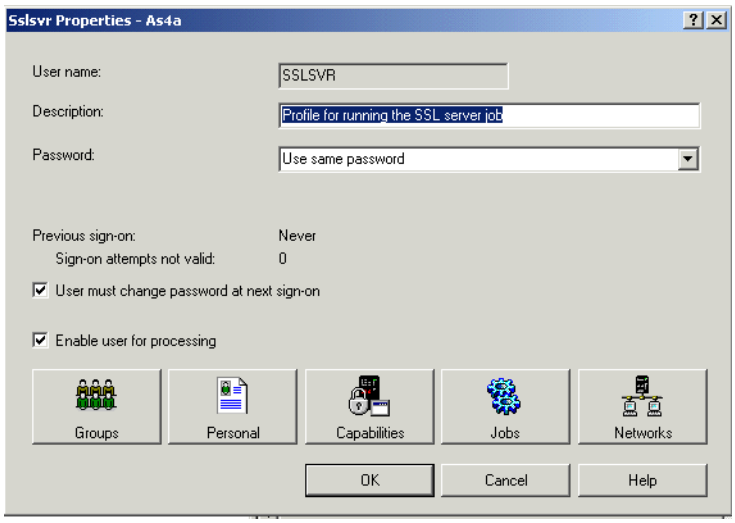


*Figure 301. Giving access to the *SYSTEM certificate store 2*

6. Click **Capabilities** and then the **Applications** tab. You will see the display in Figure 302.

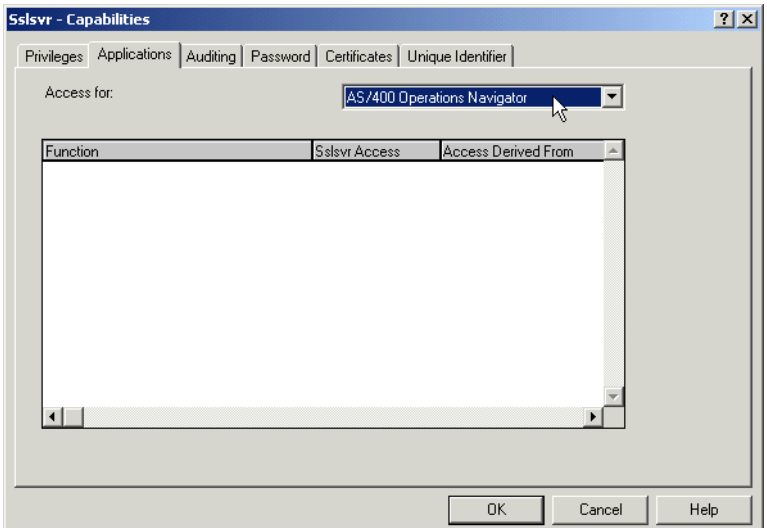


*Figure 302. Giving access to the *SYSTEM certificate store 3*

7. Open the Access for: drop-down list and select **Host applications**.
8. A list of the existing host applications will be displayed. Expand **Digital Certificate Manager (DCM)**.
9. A list of DCM options will appear including ***SYSTEM certificate store**. See Figure 303.

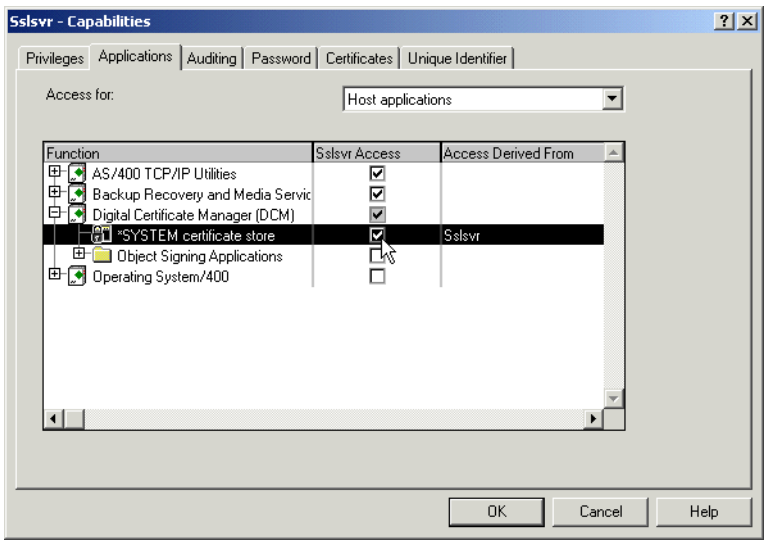


*Figure 303. Giving access to the *SYSTEM certificate store*

10. Check the box to the right of ***SYSTEM Certificate Store**.
11. Finally click **OK** to exit the Capabilities window and click **OK** again to exit the user properties window.
12. Repeat for the all the user profiles that need authority to the ***SYSTEM** certificate store.

Note

It is a bad idea to grant specific authorities to multiple user profiles for anything, because this becomes too hard to manage. It is better to create a group profile, grant the authority once to it, and give the group profile to the users that require access to the object.

Keep your authority scheme as simple as possible. Otherwise, management becomes difficult and confidence in the integrity of your scheme is hard to maintain.

It is also a bad idea to try to directly grant users authority to the *SYSTEM certificate store files. This requires giving authority to a number of directories that users have no need to access.

Appendix C. Enabling SSL for the ADMIN server instance

Through the AS/400 Tasks page, you can configure, for example the HTTP Server for AS/400, the Digital Certificate Manager, or the 4758 PCI Cryptographic Coprocessor for iSeries. The configuration of these applications requires that sensitive and confidential data is sent between the administrator's Web browser and the iSeries or AS/400 system. To securely transmit this data, you have to enable SSL for the ADMIN server instance of the HTTP Server.

The tasks to enable SSL for the ADMIN HTTP server instance are:

- Changing the ADMIN HTTP configuration.
- Assign a server certificate to the ADMIN server.
- Restarting the server to activate the configuration changes.

Note that the ADMIN server instance in V5R1 runs as an HTTP Server (powered by Apache) instance.

C.1 Changing the ADMIN server configuration

The following steps show how to enable SSL for the ADMIN server:

1. Start a Web browser and go to the AS/400 Tasks page using the URL:

   `http://servername:2001`

2. Select **IBM HTTP Server for AS/400** from the AS/400 Tasks page. The main page of the IBM HTTP Server for AS/400 is displayed (Figure 304 on page 416).

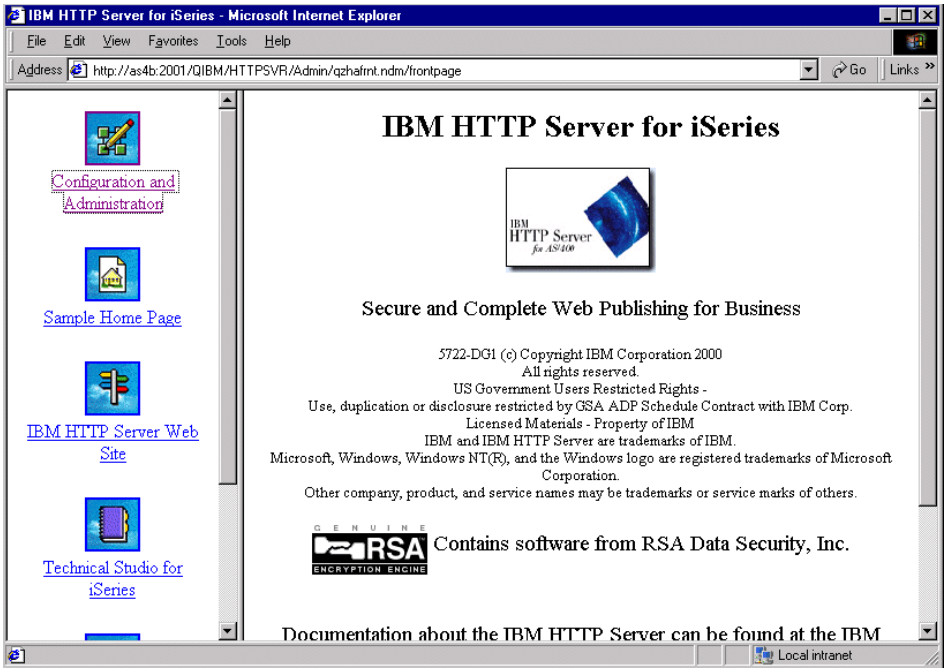


Figure 304. IBM HTTP Server for AS/400 main page

3. Click **Configuration and Administration** in the left pane of the window to open the HTTP Server configuration and administration tasks.
4. Click the **Configurations** tab and select **ADMIN** from the Configuration for server list. You see the display shown in Figure 305.

Figure 305. HTTP Server (powered by Apache) ADMIN configuration

5. Click **ADMIN global settings** in the navigation pane.
6. Scroll down the right pane until you see the Configuration Files section, as shown in Figure 306 on page 418.

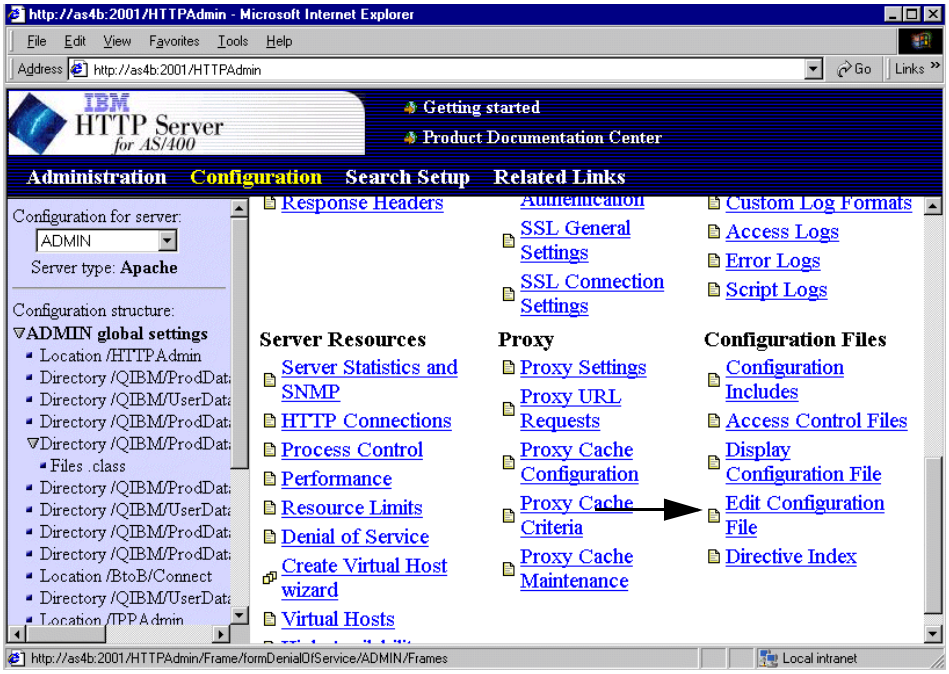


Figure 306. Configuration Files section

7. Click **Edit Configuration File** to open the ADMIN server configuration file.
8. When the configuration file has opened, scroll down to the bottom and copy the following text into the clipboard:

```
#-----------------------------------------------------
# The following directives should be added to
# /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf
# and uncommented in order to enable SSL for ADMIN.
#-----------------------------------------------------
# LoadModule ibm_ssl_module /QSYS.LIB/QHTTPSVR.LIB/QZSRVSSL.SRVPGM
# Listen 2001
# Listen 2010
# SetEnv HTTPS_PORT 2010
# <VirtualHost *:2010>
#   SSLEnable
#   SSLAppName QIBM_HTTP_SERVER_ADMIN
# </VirtualHost>
```

9. Click **Cancel** to exit from the IBM-supplied configuration file.
10. Scroll down the navigation pane.

Figure 307. ADMIN server navigation pane

11. Click **Include /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf** in the navigation pane, as shown in Figure 307. The options in the right pane change and list the available options for the ADMIN server user configuration file.

12. Scroll down on the right pane until you see the Configuration Files section and click **Edit Configuration File**. You see the display shown in Figure 308 on page 420.

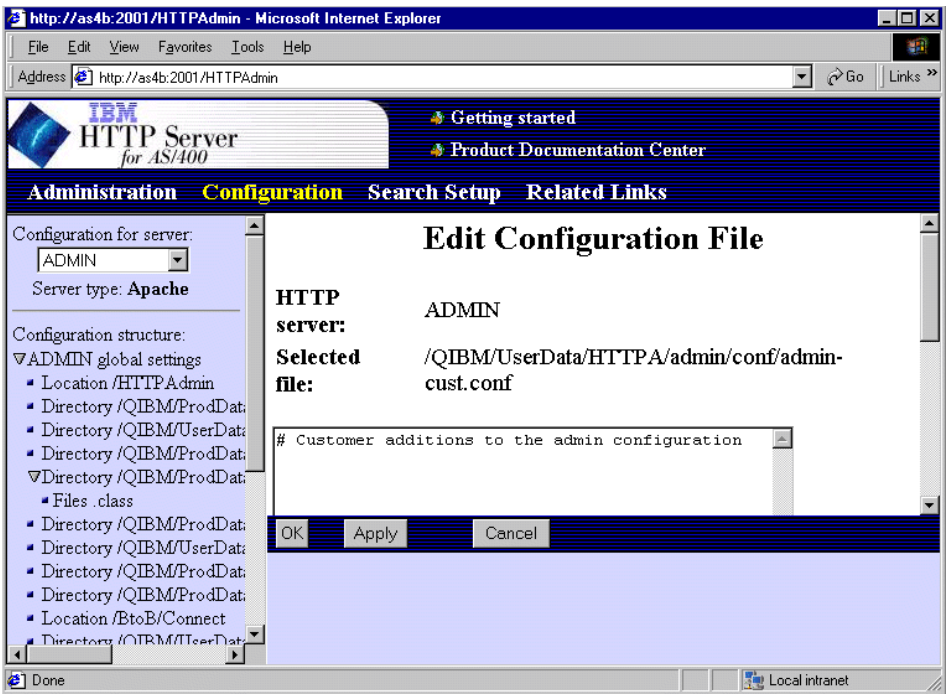


Figure 308. Editing the ADMIN server user configuration file

13. Paste the directives from the clipboard into the configuration file.
14. Remove the leading # character from the lines containing directives as shown in Figure 309.

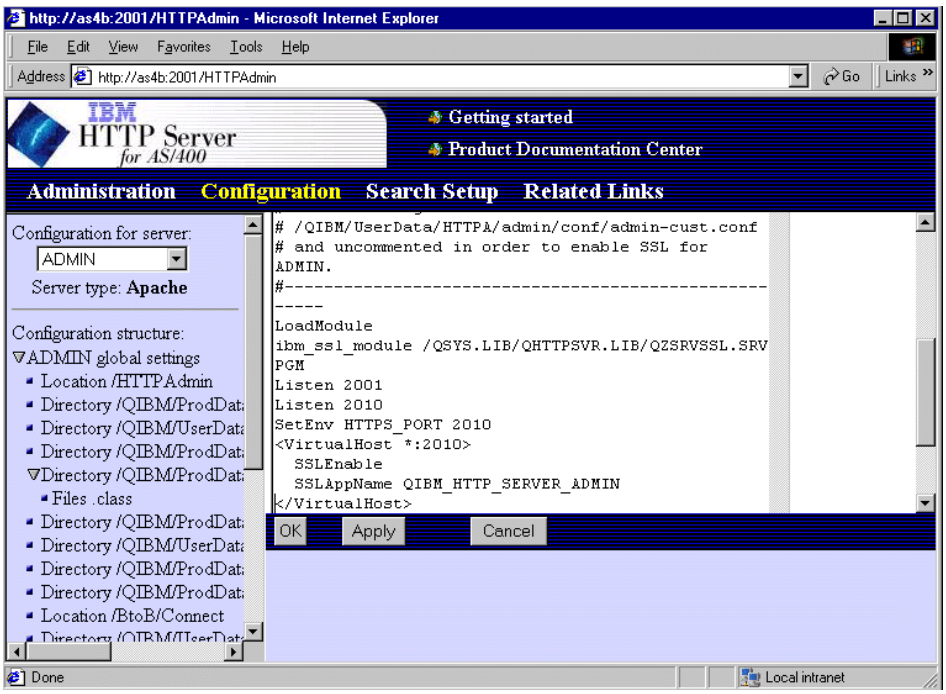


Figure 309. Editing the ADMIN server user configuration file with added directives

Note that the comment section at the beginning still contains the leading # character.

15. Click **Apply** to save your changes.

The application name QIBM_HTTP_SERVER_ADMIN also has to be registered in the Digital Certificate Manager (DCM) in order to assign a certificate to the ADMIN server instance. If you were *not* running the original server's ADMIN instance with SSL in a release prior to V5R1, you have to enter the following command on a command line to register the application name in DCM:

```
call qhttpsvr/qzhapreg parm('RegisterAppName' 'QIBM_HTTP_SERVER_ADMIN')
```

Note that it will not cause any error when running the command even if the application name is already registered in DCM.

Important

Do not restart the ADMIN server yet, unless you had enabled SSL for the ADMIN server instance in a previous release. The server certificate must be assigned first; otherwise, the server will fail during restart.

C.2 Assigning a server certificate to the ADMIN server

Every SSL-enabled server application needs to have a digital server certificate assigned to it. To assign a server certificate to the ADMIN HTTP server, perform the following steps:

1. From the AS/400 Tasks page, click **Digital Certificate Manager**.
2. Click **Select a Certificate Store** in the navigation pane.

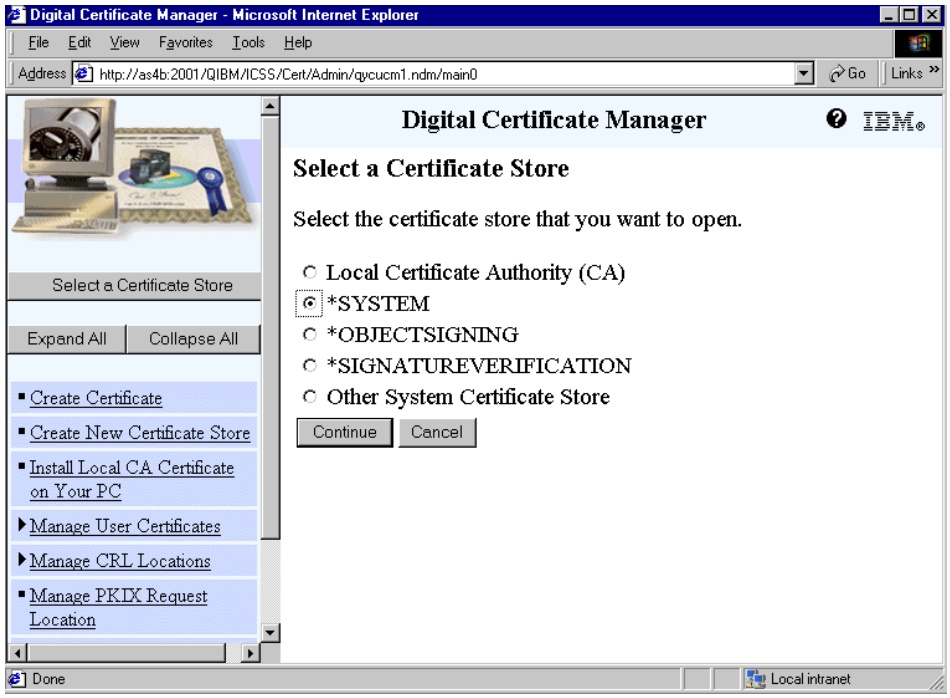


Figure 310. Digital Certificate Manager - Select a Certificate Store window

3. Select the ***SYSTEM** certificate store as shown in Figure 310 and click **Continue**.
4. Enter the password for the *SYSTEM certificate store and click **Continue**.

5. Click **Fast Path** and then **Work with server applications** in the navigation pane.

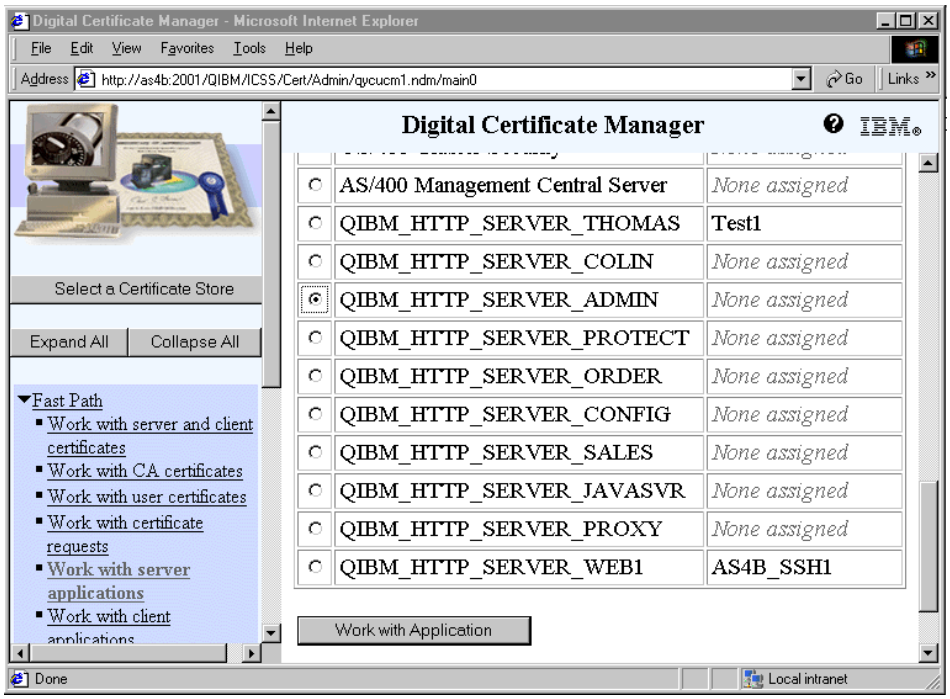


Figure 311. Digital Certificate Manager - Work with server application window

6. Select the **QIBM_HTTP_SERVER_ADMIN** application as shown in Figure 311 and click **Work with Application**.

   Note that there is probably already a server certificate assigned to the ADMIN server instance, if you enabled SSL for this server in a previous release. See Figure 312 on page 424.

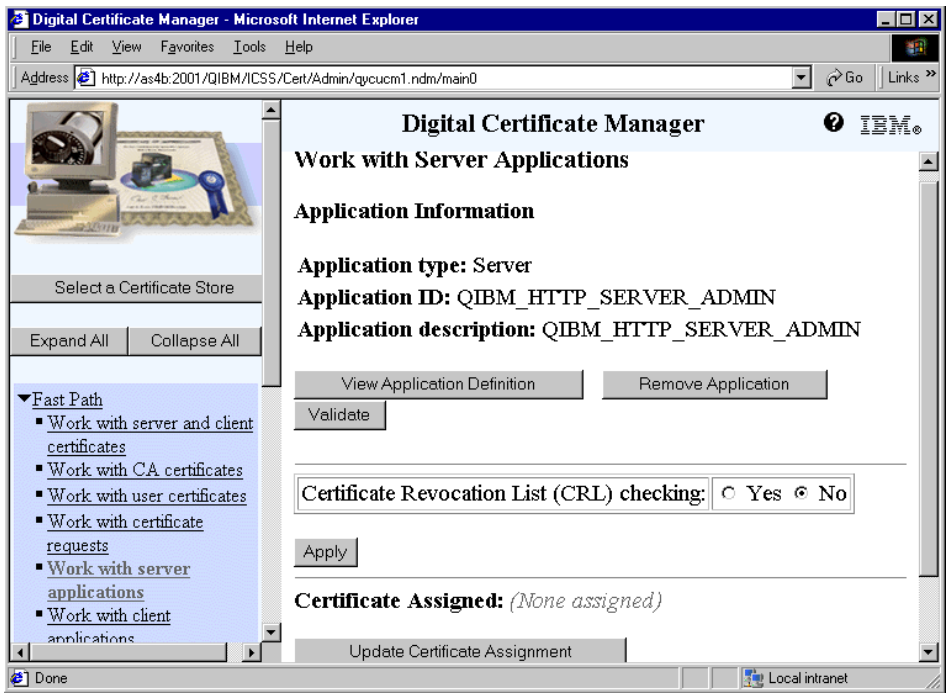


Figure 312. Digital Certificate Manager - Application Information window

7. From the Application Information window shown in Figure 312, click **Update Certificate Assignment**.

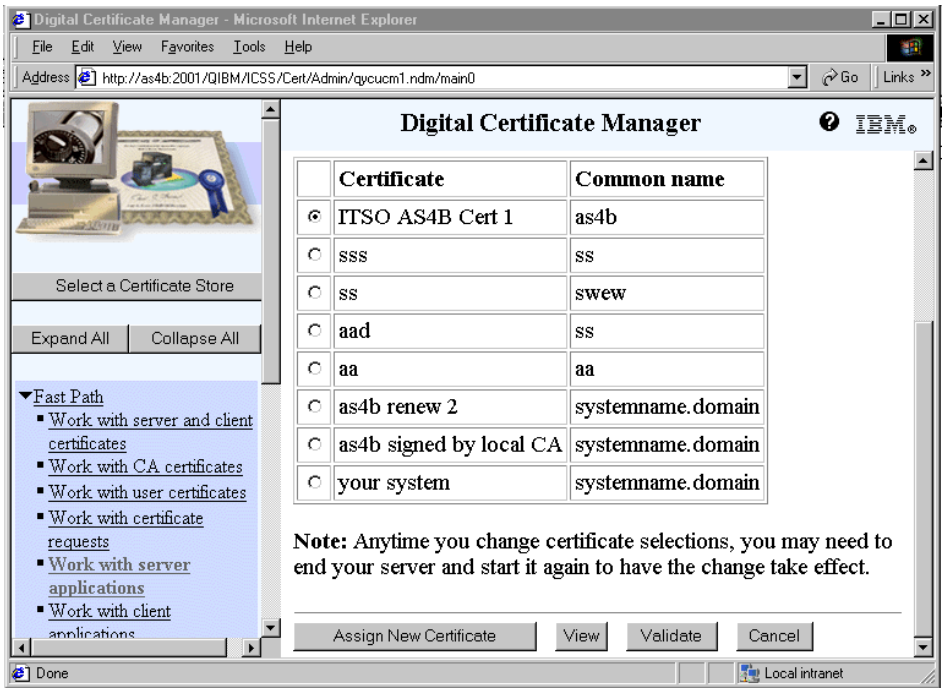


Figure 313. Digital Certificate Manager - Update Certificate Assignment window

8. Select the server certificate you want to assign to the ADMIN server instance as shown in Figure 313 and click **Assign New Certificate**. A completion message confirms that the certificate was assigned successfully.

For more information on assigning certificates to an application, refer to the document *Digital certificate management* found in the iSeries Information Center by clicking **Security->Digital certificate management**.

C.3 Activating the configuration changes

After the ADMIN server instance is configured for SSL and a server certificate is assigned, the ADMIN server needs to be stopped and started. Restarting the server instance will not activate the changes.

You can stop and start the server through the command line interface using the following commands:

```
ENDTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
```

C.4 Deactivating SSL for the ADMIN server without using the GUI

Usually the Web-based configuration interface, started through the AS/400 Tasks page, is used to configure HTTP server instances. This configuration interface is controlled by the HTTP ADMIN server instance. In some cases where, for example, the server certificate used by the ADMIN server has expired or the certificate store was deleted, the ADMIN server will fail to start with SSL initialization errors. In this case, the configuration interface via the AS/400 Tasks page will not work and you have to manually disable SSL support for the ADMIN server to recover from this error.

The following steps explain how to deactivate SSL support without the configuration interface:

1. Sign on to the iSeries or AS/400 system with a user profile having enough authority to edit the server configuration files; preferably with *ALLOBJ special authority.
2. On the command line, enter the following command to edit the user part of the ADMIN server configuration:

   ```
   EDTF STMF('/QIBM/UserData/HTTPA/admin/conf/admin-cust.conf')
   ```

   Figure 314 shows the SSL part of the configuration.

```
 Edit File: /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf
 Record :      1   of     14 by  10                  Column :    1     64 by 126
 Control :

CMD
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+..
0....+.
    ************Beginning of data**************
   # Customer additions to the admin configuration
   #-------------------------------------------------------
   # The following directives should be added to
   # /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf
   # and uncommented in order to enable SSL for ADMIN.
   #-------------------------------------------------------
   LoadModule ibm_ssl_module /QSYS.LIB/QHTTPSVR.LIB/QZSRVSSL.SRVPGM
    Listen 2001
    Listen 2010
    SetEnv HTTPS_PORT 2010
    <VirtualHost *:2010>
     SSLEnable
     SSLAppName QIBM_HTTP_SERVER_ADMIN
    </VirtualHost>
     ************End of Data********************
 F2=Save  F3=Save/Exit   F12=Exit   F15=Services   F16=Repeat find   F17=Repeat change
F20=Right
             (C) COPYRIGHT IBM CORP. 1980,  2000.
```

Figure 314. Customized ADMIN server configuration with SSL enabled

3. Comment out the configuration required for SSL by placing a # character in front of the corresponding directive as shown in Figure 315 on page 428.

```
 Edit File: /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf
 Record :      1   of     14 by  10                  Column :    1     64 by 126
 Control :

CMD
....+....1....+....2....+....3....+....4....+....5....+....6....+....7....+....8....+..
0....+..
    ************Beginning of data**************
   # Customer additions to the admin configuration
   #-------------------------------------------------------
   # The following directives should be added to
   # /QIBM/UserData/HTTPA/admin/conf/admin-cust.conf
   # and uncommented in order to enable SSL for ADMIN.
   #-------------------------------------------------------
   # LoadModule ibm_ssl_module /QSYS.LIB/QHTTPSVR.LIB/QZSRVSSL.SRVPGM
   # Listen 2001
   # Listen 2010
   # SetEnv HTTPS_PORT 2010
   # <VirtualHost *:2010>
   #  SSLEnable
   #  SSLAppName QIBM_HTTP_SERVER_ADMIN
   # </VirtualHost>
     ************End of Data********************
 F2=Save  F3=Save/Exit   F12=Exit   F15=Services   F16=Repeat find   F17=Repeat change
F20=Right
             (C) COPYRIGHT IBM CORP. 1980,  2000.
```

Figure 315. Customized ADMIN server configuration with SSL disabled

4. Press F3 to save the configuration and start the ADMIN server instance again with the command:

   ```
   STRTCPSVR SERVER(*HTTP) HTTPSVR(*ADMIN)
   ```

If necessary you can start the Web-based configuration interface and enable SSL again after the ADMIN server instance is up and running.

Note

The SSL-deactivation steps shown in this section describe how to deactivate SSL for the ADMIN instance with OS/400 V5R1. Use the command `WRKHTTPCFG *ADMIN` to deactivate SSL for the ADMIN instance in releases prior to V5R1.

Appendix D. Creating a local Certificate Authority

A Certificate Authority (CA) is a central administrative entity that can issue digital certificates to users and servers. The Certificate Authority *signs* certificates with its private key to validate their authenticity. A CA can be either a publicly available entity, such as VeriSign, or it can be a privately created entity, such as a private intranet or local CA. Digital Certificate Manager (DCM) allows you to use both types of CAs to request and install certificates.

When you use DCM to create a local CA for your organization, you can use the CA to issue certificates for server and client applications, object signing, and users on your system. When the Certificate Authority issues a user certificate, DCM automatically associates the certificate with the appropriate OS/400 user profile. This ensures that the access and authorization privileges for the certificate are the same as those for the owner's user profile.

Follow these steps to create a local Certificate Authority using the redesigned DCM interface:

1. Log on to the AS/400 Tasks page. Enter the URL:

   `http://AS400:2001`

2. Sign on with your user ID and password:

   Note that the user profile needs to have *ALLOBJ and *SECADM special authorities.

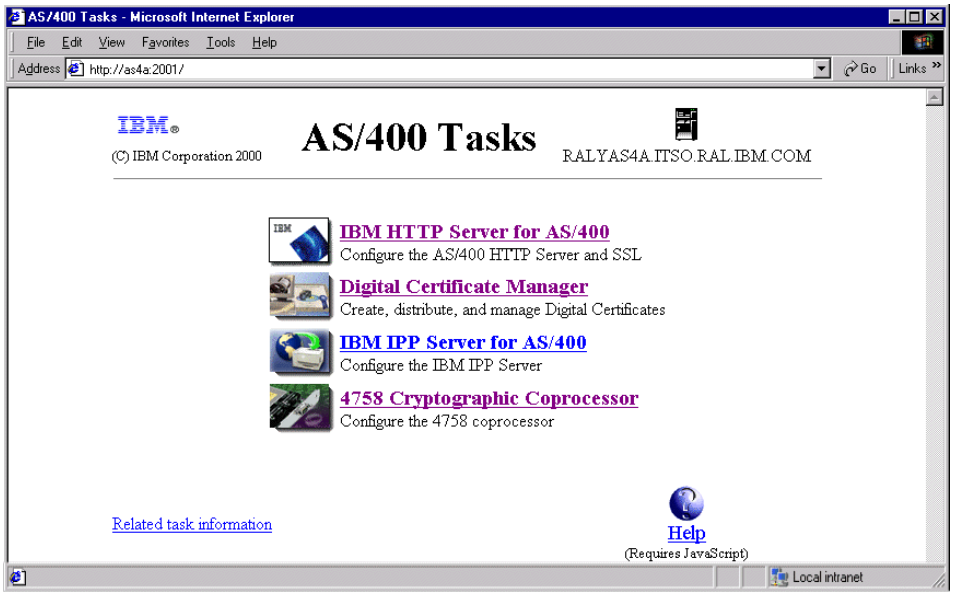


Figure 316. AS/400 Tasks Page

3. Click **Digital Certificate Manager**.

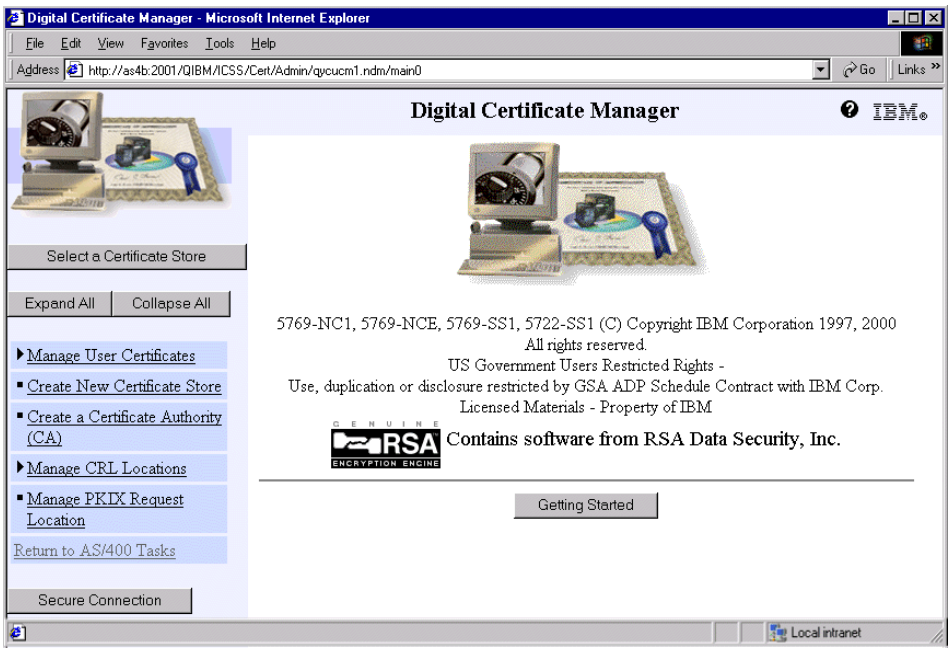


Figure 317. Create a Certificate Authority

4. Click **Create a Certificate Authority (CA)**.

   If you have a 4758 Cryptographic Coprocessor installed and configured, you will be presented first with a selection window to specify where to store the Certificate Authority's private key. An example is shown in Figure 318. Otherwise the window in Figure 319 on page 432 is shown.

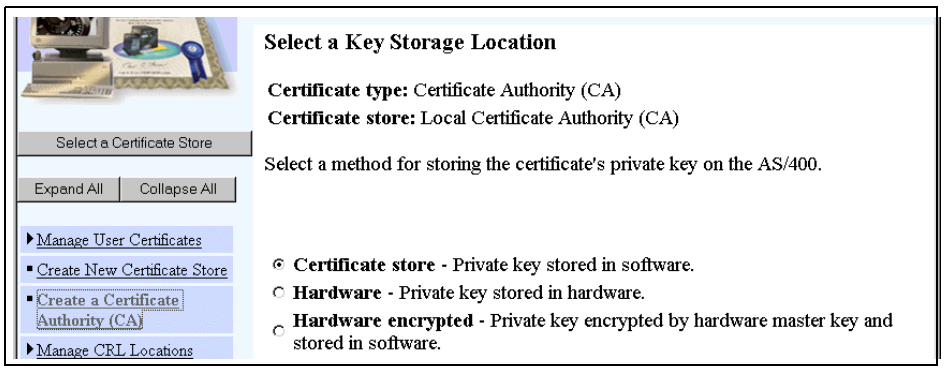


Figure 318. Select a Key Storage Location window

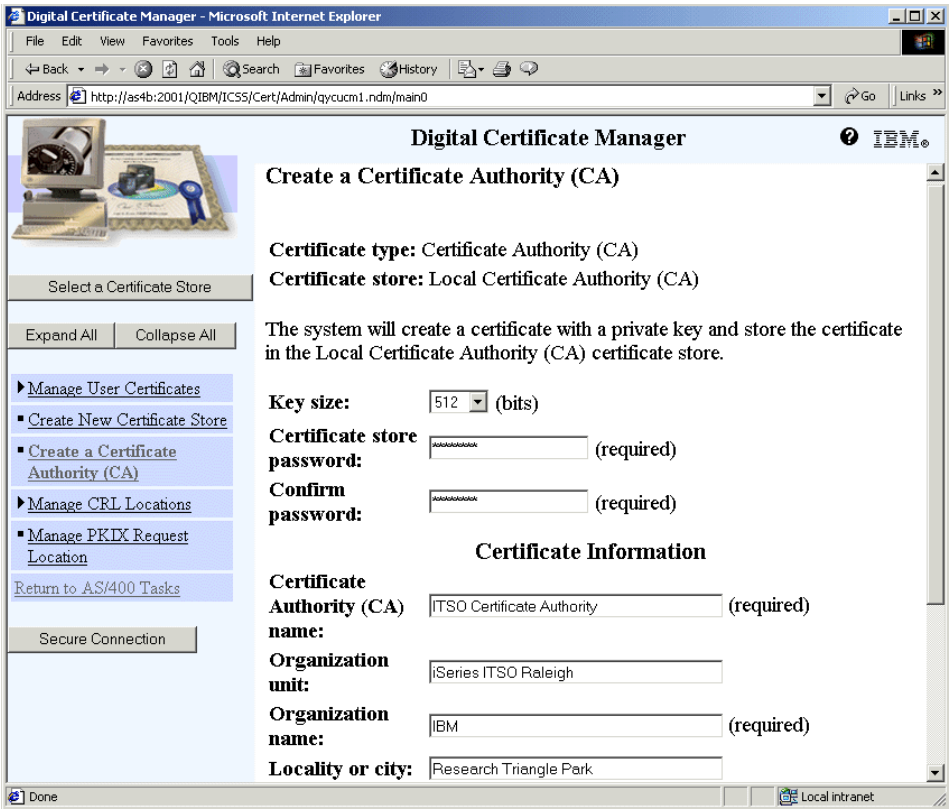


Figure 319. Create Certificate Authority Form

5. Complete the Create a Certificate Authority (CA) window and click **Continue**.

   Table 20 shows the fields in the Create a Certificate Authority (CA) window and the values used in this setup.

Table 20. Create a Certificate Authority (CA) window fields

Field	Value
Key size	512
Certificate store password	password
Confirm password	password
Certificate Authority name	ITSO Certificate Authority
Organization unit	iSeries ITSORaleigh

Field	Value
Organization name	IBM
Locality or city	Research Triangle Park
State or province	North Carolina
Country	US
Validity period of Certificate Authority	2000

Digital Certificate Manager processes the form and creates the following files in the directory `/QIBM/UserData/ICSS/Cert/CertAuth`:

- CA.TXT: Contains the CA certificate in Base64 encoded form
- DEFAULT.KDB: Contains the private key and the CA certificate
- DEFAULT.POL: Is the CA policy file
- DEFAULT.RDB: Is the CA's request database.

Note

Note that this directory in releases prior to V5R1 also contained a stashed password file (DEFAULT.STH). This file contains the CA certificate store password in stashed form and was used by system functions to access the database. In V5R1 and through PTFs in V4R5, the stashed password file no longer exists. Instead the stashed password is stored in an internal system object not accessible by any user program.

In the `/QIBM/UserData/ICSS/Cert/Download/CertAuth` directory is the CA.CACRT file, the binary form of the CA certificate.

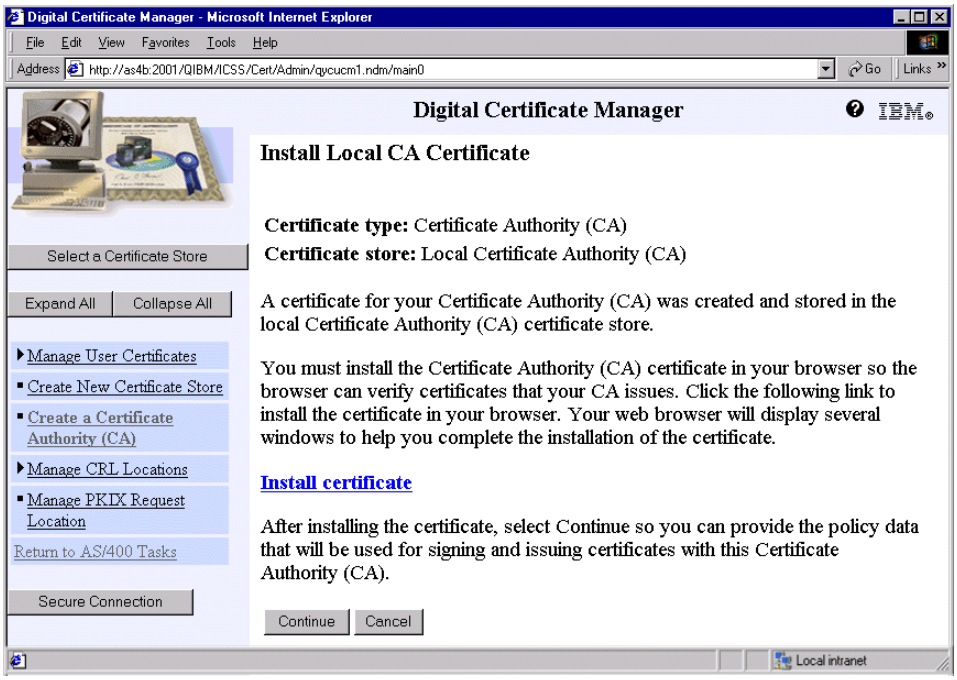


Figure 320. Install Local CA Certificate window

The Install Certificate link allows you to install the CA certificate on your browser.

6. Click the **Install certificate** link to receive the CA certificate and then click **Continue**. Or you may just click **Continue** if you do not need the CA certificate on the PC you are performing the configuration with.

The Certificate Authority (CA) Policy Data window is displayed.

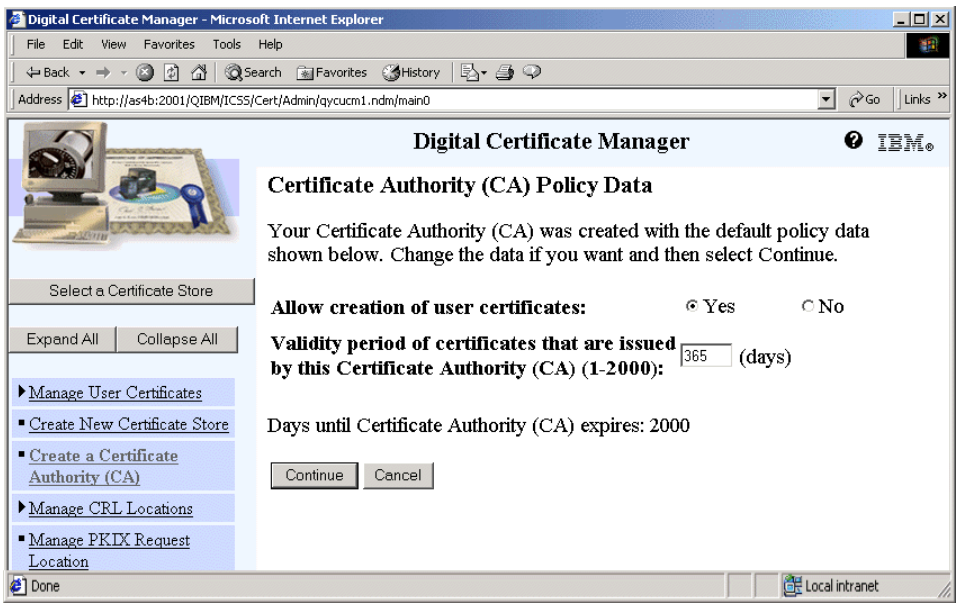


Figure 321. Certificate Authority (CA) Policy Data window

7. Select **Yes** to Allow creation of user certificates and click **Continue**.

   You have now created a local CA on your system and can use it to issue certificates for server and client applications, object signing, and users.

> **Note**
>
> When you create a Certificate Authority (CA) with Digital Certificate Manager, you can specify the policy data for the CA. The policy data for a (CA) describes the signing privileges that it has. The policy data determines whether the CA can issue and sign user certificates and how long certificates that the CA issues are valid.

You see a confirmation message that the policy data has been accepted (see Figure 322 on page 436.)

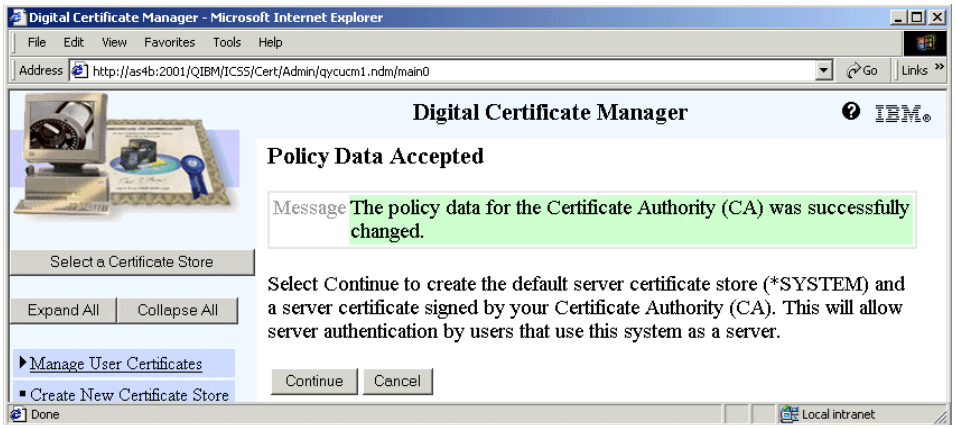


Figure 322. Policy Data Accepted window

8. Click **Continue**.

   The next step is to create a server certificate that secure server and client applications can use during the SSL handshake.

9. Complete the Create a Server or Client Certificate form (Figure 323). Table 21 shows the values you have to enter in each field in the window.

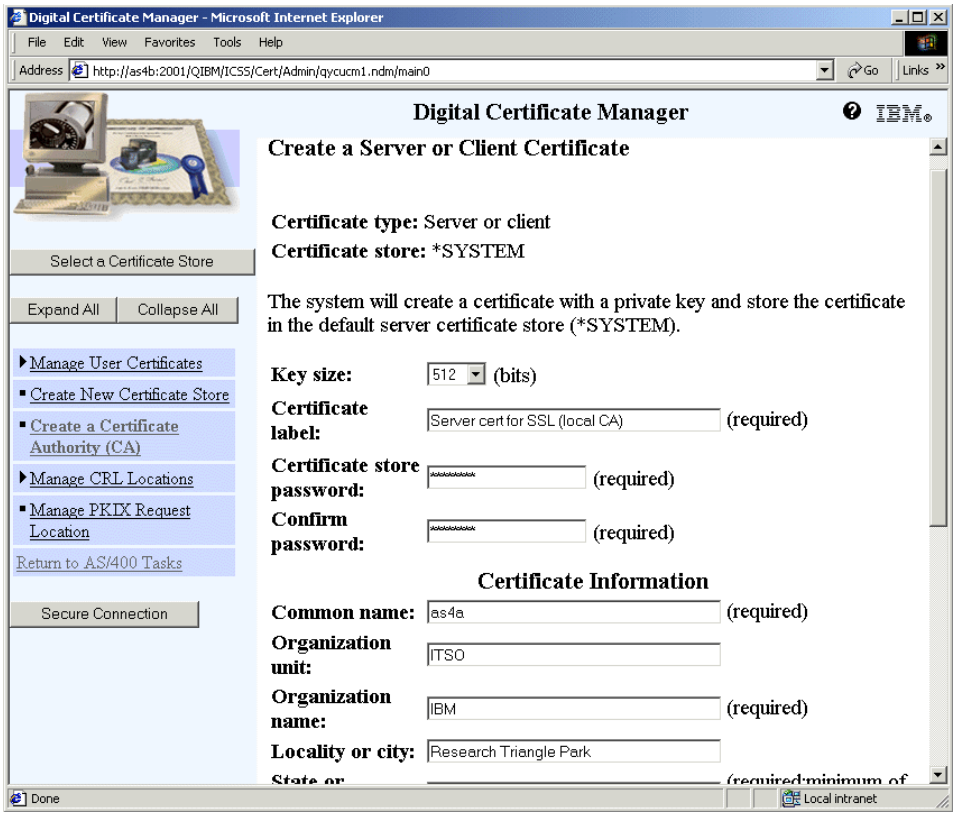


Figure 323. Create a Server of Client Certificate window

Table 21 shows the fields in the Create a Server or Client Certificate window and the values used in this setup.

Table 21. Create a Server or Client Certificate window fields

Field	Value
Key size	512
Certificate label	Server cert for SSL (local CA)
Certificate store password	password
Confirm password	password
Server name	as4b
Organization unit	ITSO
Organization name	IBM

Field	Value
Locality or city	Research Triangle Park
State or province	North Carolina
Country	US
Subject Alternative Name fields	blank

10. Click **Continue**.

    DCM creates the system certificate in the *SYSTEM certificate store. The *SYSTEM certificate store consists of the following files in the `/QIBM/UserData/ICSS/Cert/Server` directory:

    - **DEFAULT.KDB**: Contains server and client certificates with their private keys
    - **DEFAULT.RDB**: Is the certificate request database

    DCM displays the next window to select applications that will use this server certificate. At the top of the window a confirmation message is displayed that the server certificate has been created.

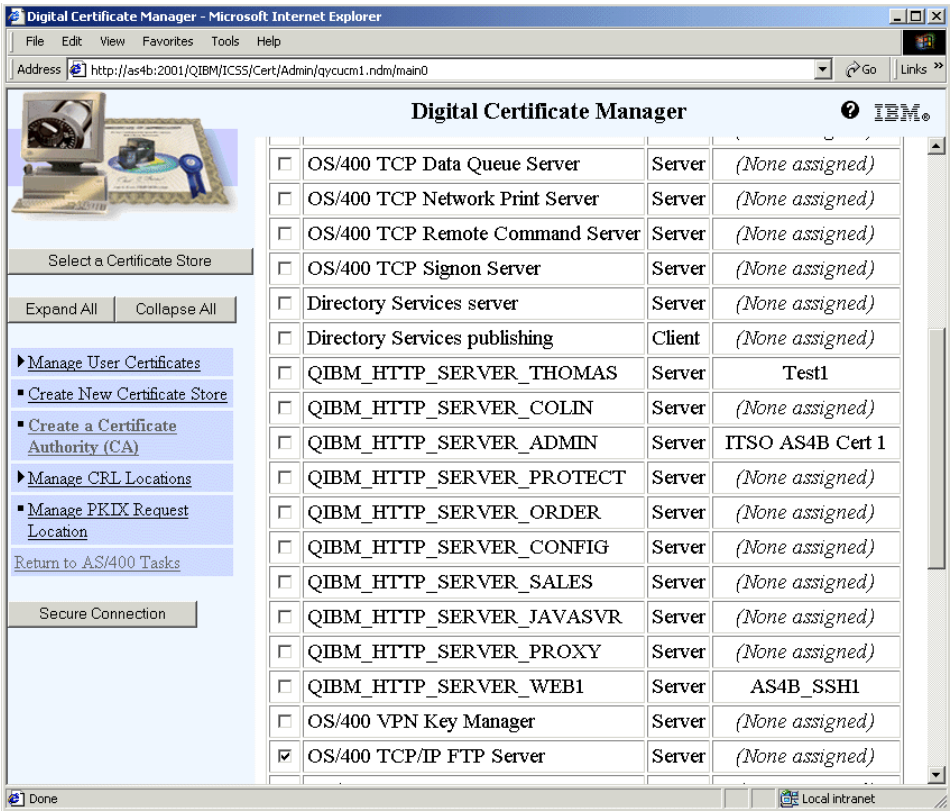


Figure 324. Select applications that will use this certificate

11. Select the server applications that will use the new server certificate.
12. Click **Continue** at the bottom of the window.

    A confirmation message is displayed saying that the selected applications will use the new server certificate. See Figure 325 on page 440.

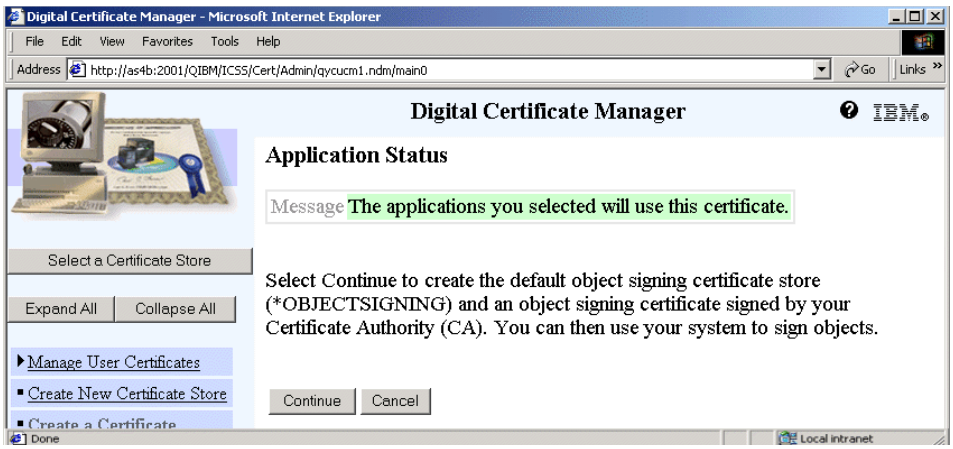


Figure 325. Application Status window

The next steps are new in V5R1 and will create the object signing certificate store.

13. Click **Continue**.

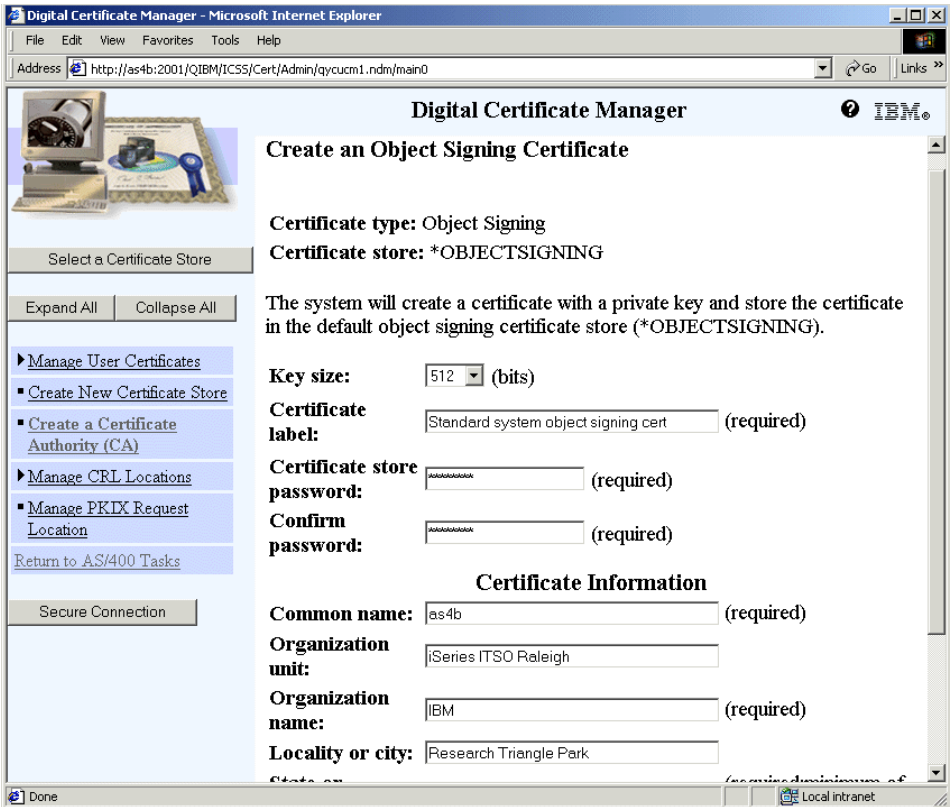


Figure 326. Create an Object Signing Certificate window

Enter the certificate's RSA key length, object signing certificate store password, and a label for the new object signing certificate. Then enter the certificate's subject information as previously done for the server certificate.

14. Click **Continue**.

The Select Applications window is displayed, which contains a confirmation message about the creation of the object signing certificate. If you used object signing before on this system, a list of available object signing applications is also shown. You can then select the applications you want to use the new certificate. In a new system environment, no applications are shown.

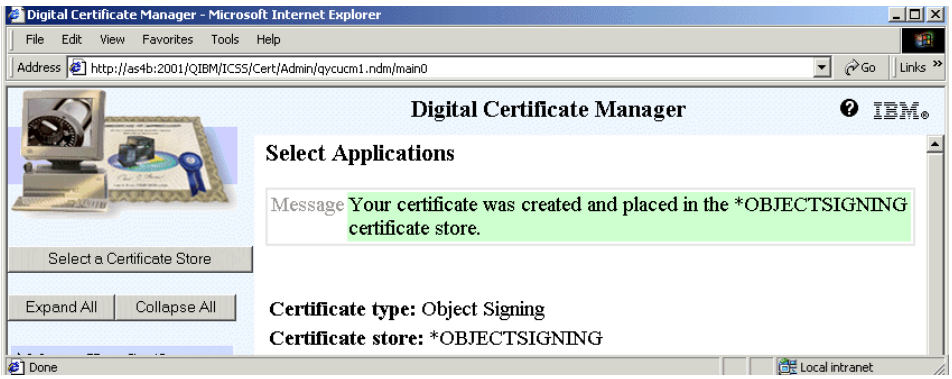


Figure 327. Select Applications window

15. Click **Continue**.

Now you have to select all the applications that will trust the newly created Certificate Authority. The list contains all registered client applications as well as the server applications that support client authentication.

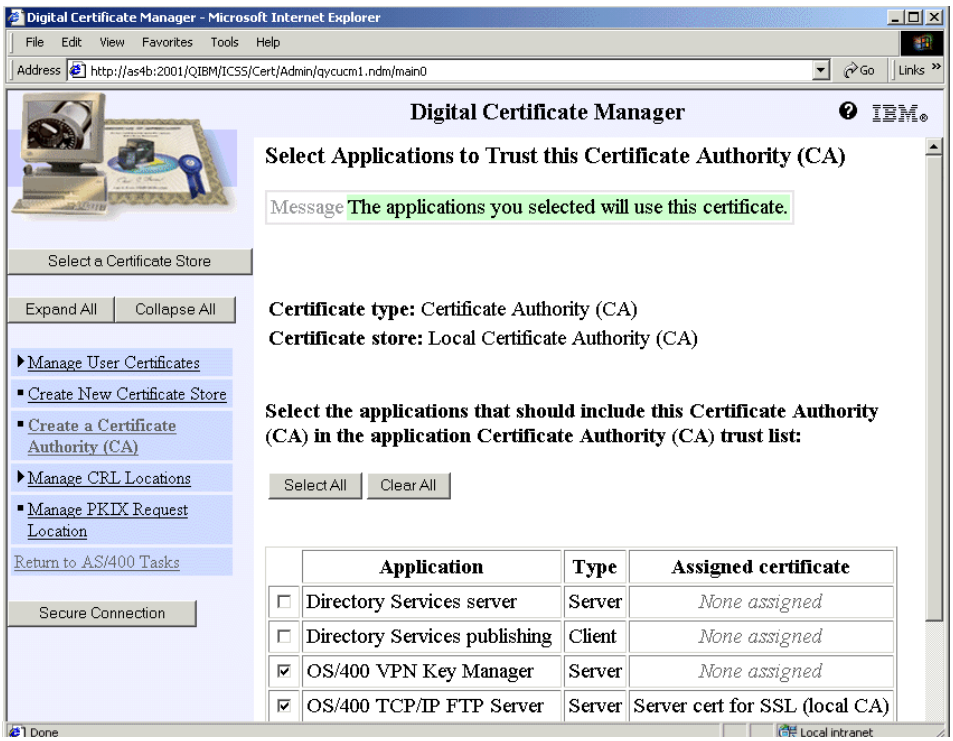


Figure 328. Select Applications to Trust this Certificate Authority (CA) window

16. Select the applications you want to trust the new CA and click **Continue**.

    A final confirmation message shows that the Certificate Authority setup process is completed (shown in Figure 329).

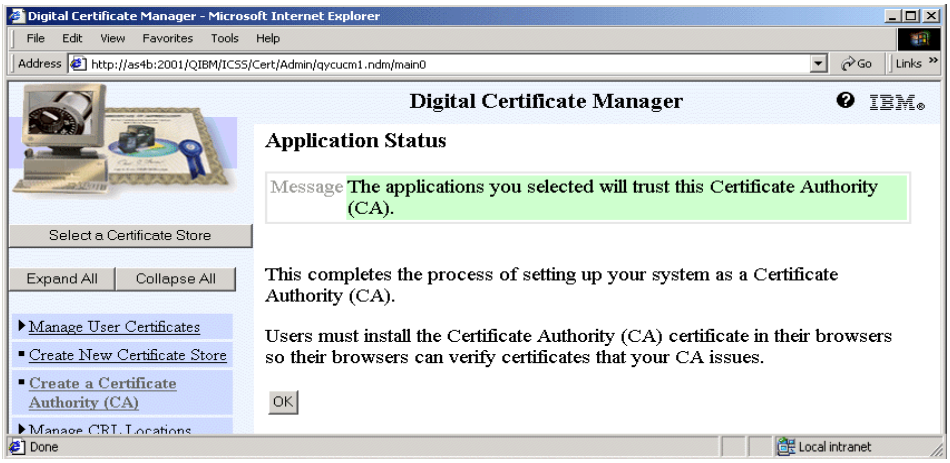


Figure 329. Application Status window

17. Click **OK** to complete the setup.

> **Note**
>
> The first time the local CA and *SYSTEM certificate store are created, there is no default certificate assigned to the certificate store as shown in Figure 330 on page 444. As long as all SSL applications, whether system-provided or user-written, explicitly specify a certificate when establishing a SSL session, the default certificate setting is not important. But there are also applications, mostly user-written ones, that may not specify a certificate and then the default certificate is chosen. For this reason it is recommended that you set a default certificate label for the *SYSTEM certificate store.

18. Click **Select a Certificate Store** on the DCM navigation pane.
19. Select the ***SYSTEM** certificate store and click **Continue**.
20. Enter the certificate store password as specified in Table 21 on page 437 and click **Continue**.
21. Click **Fast Path**.
22. From the Fast Path options, click **Work with server and client certificates**.

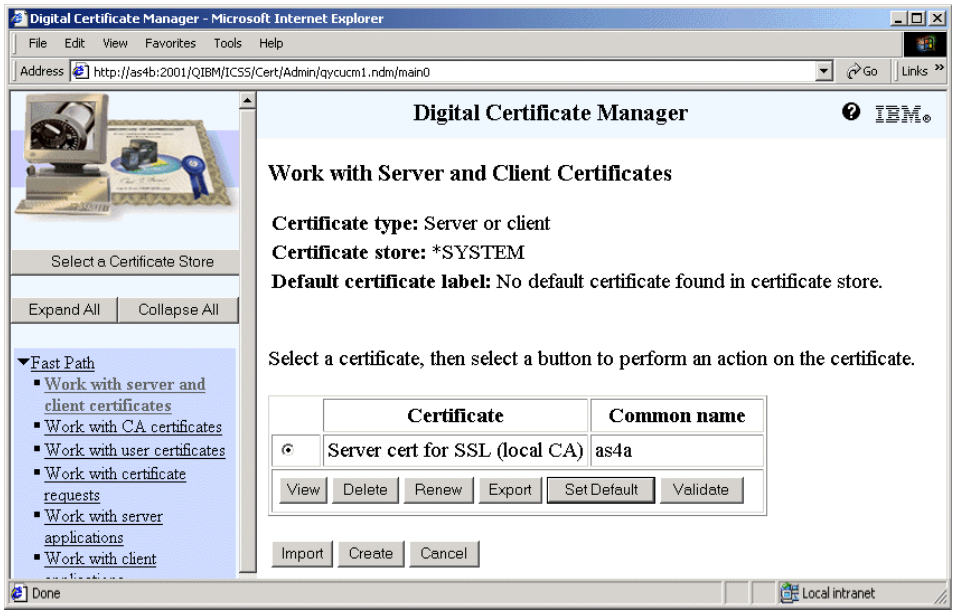


Figure 330. Work with Server and Client Certificates window - no default certificate assigned

23. To assign a default certificate label to the *SYSTEM certificate store, select the server certificate you want to use as the default certificate and click **Set Default**.

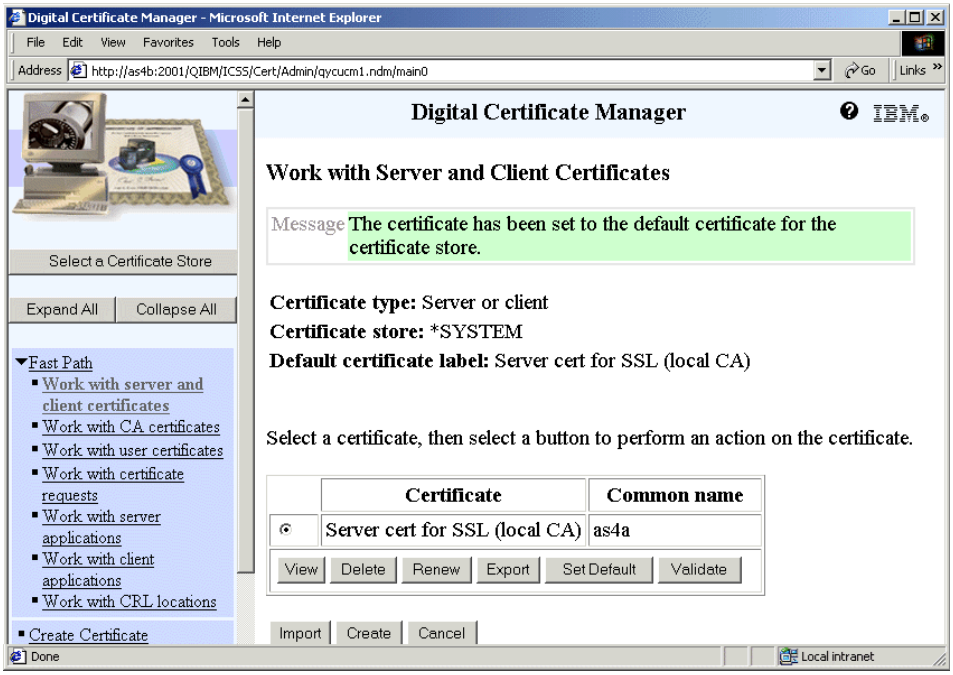


Figure 331. Work with Server and Client Certificates window: Default certificate label set

24. Click **Cancel** to complete the tasks.

Appendix E. Certificate import/export interoperability tests

While we were writing this redbook, we performed some interoperability testing by importing and exporting certificates to and from various platforms. You may experience different results when using other versions of the various operating system platforms and key management products. In our case, we mostly used local CA functions to create the certificates used during interoperability testing.

E.1 Import interoperability test results

The following list shows the results we received when we imported certificates from various platforms into the OS/400 DCM *SYSTEM certificate store:

- **Windows 2000 Server CA services->OS/400 DCM V5R1**

  We created the certificate, filling in the format directly on the Windows 2000 Server. We selected the box to create the pair of keys and to mark them as exportable. To retrieve the pair of keys and the digital certificate, we had to install them in our Internet Explorer as an intermediate step, and then export it while disabling the strong encryption. On the certificate store, we saw the server or client certificate with a label of hexadecimal characters as shown in Figure 35 on page 64.

- **AIX 4.33 SNAKEYMAN->OS/400 DCM V4R5**

  We had to transport the whole key database file (file.kdb) via FTP in binary format, and then open it with the DCM. We were then able to manage the certificate and eventually export it to a file and import it into the *SYSTEM certificate store.

- **AIX 4.33 SNAKEYMAN->OS/400 DCM V5R1**

  We exported a certificate signed by a well-known CA as a PKCS12 file and chose the weak encryption. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

- **S/390 V2R10 GSKYMAN->OS/400 DCM V4R5**

  We exported a certificate signed by a well-known CA as a PKCS12 file. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

- **S/390 V2R10 GSKYMAN->OS/400 DCM V5R1**

  We exported a certificate signed by a well-known CA as a PKCS12 file. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

- **PCOM 4.3 or CA express IKEYMAN->OS/400 DCM V4R5**

  We exported a certificate signed by a well-known CA as a PKCS12 file. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

- **PCOM 4.3 or CA express IKEYMAN->AS/400 DCM V5R1**

  We exported a certificate signed by a well-known CA as a PKCS12 file. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

- **PCOM 5.0 or CA express V5R1 IKEYMAN->AS/400 DCM V4R5**

  We had to transport the whole key database file (file.kdb) via FTP in binary format, and then open it with the DCM. We were then able to manage the certificate and eventually export it to a file and import it into the *SYSTEM certificate store.

- **PCOM 5.0 or CA express V5R1 IKEYMAN->AS/400 DCM V5R1**

  We exported a certificate signed by a well-known CA as a PKCS12 file and chose the weak encryption. We moved the file via FTP with binary format. Once imported, the certificate kept its original label.

E.2 Export interoperability test results

The following list shows the results when we exported certificates from the OS/400 DCM *SYSTEM certificate store and imported them on another platform:

- **AS/400 DCM V5R1->AIX 4.33 SNAKEYMAN**

  We had to transport the whole key database file (file.kdb) via FTP in binary format, and then open it with the SNAKEYMAN. We were then able to manage the certificate.

- **AS/400 DCM V5R1->S/390 V2R1 GSKYMAN**

  We had to create the certificate without extensions (subject alternative names) and export it by selecting V4R5 or V4R4 as the target release. That means that the GSKYMAN V2R1 supports PKCS#12 V1.

- **AS/400 DCM V5R1->S/390 V2R1 RACDCERT**

  We had to export the certificate by selecting V5R1 as the target release. That means that the RACF Digital Certificate V2R1 supports PKCS#12 V3. We were able to import the certificate with the subject alternative names, but we could not display them.

- **AS/400 DCM V5R1->PCOM 4.3 or CA express**

  We had to export the certificate by selecting V4R5 or V4R4 as the target release. That meant that its IKEYMAN product supported PKCS#12 V1.

- **AS/400 DCM V5R1->PCOM 5.0 or CA V5R1**

  We had to export the certificate by selecting V4R5 or V4R4 as the target release. That meant that its IKEYMAN product supported PKCS#12 V1. The import operation was successfully completed, but we were not able to display or manage the certificate under the list of personal certificates.

Appendix F. Publishing a CRL to an OS/400 LDAP server

In OS/400 V5R1, the Certificate Revocation List (CRL) support provides certificate validation by accessing a CRL located on a Certificate Authority's Lightweight Directory Access Protocol (LDAP) server. For fast connections from your iSeries or AS/400 server to the CA's server, the CRL checking usually does not have a large impact on performance, for example, during the SSL handshake. But when using slow connections or when network congestions occur, the CRL checking via the CA's LDAP server can introduce delays. For this reason you may want to replicate the CA's CRL to a local LDAP server hosted on the iSeries or AS/400 server. This appendix describes the necessary steps to set up the LDAP server environment on the iSeries or AS/400 server. It also contains the required steps to publish a CRL to a local LDAP server. OS/400 operating system, DCM, and basic LDAP knowledge is assumed.

F.1 Planning

Before you start with the actual implementation, you need to gather certain information. First, you need to find out how your CA publishes CRLs. This varies depending on the CA. Some CAs make the CRLs available for download on their Web sites. For example, VeriSign CRLs can be downloaded from `http://crl.verisign.com`. Some other CAs will allow you to transfer CRLs using FTP. Once you know how to obtain the CRL, you need to find out how often the CRL will be updated by the CA. Even if a CRL contains information when the next CRL will be published, you have no easy access to this information without writing your own applications. Note that the CRL has to be in binary format (Distinguished Encoding Rule or DER format).

In the second step you need to find out whether the certificates issued by this CA contain CRL Distribution Point extensions. CRL Distribution Points define the search path within the LDAP directory for finding the CRL. If the CA does not support CRL Distribution Points, the default search path is the certificate issuer's distinguished name (DN). To find out what the issuer's DN is, you can use the Digital Certificate Manager (DCM) and display, for example, a server certificate as shown in Figure 332 on page 452.

Issuer:

Common name	ITSO Raleigh Root CA
Organization unit	iSeries ITSO (T.Barlen)
Organization name	IBM
Locality or city	
State or province	
Zip or postal code	
Country	US

Figure 332. Issuer's DN

In this example, the issuer's DN information in X.509 format would be:

```
CN=ITSO Raleigh Root CA, OU=iSeries ITSO (T.Barlen), O=IBM, C=US
```

Later in this appendix, when setting up the LDAP server, you need this information. Remember, when CRL Distribution Points are used, you have to use the information that they provide.

F.2 Configuration summary

The following list summarizes the steps required to use a local LDAP server for CRL checking:

1. Perform the basic configuration of the LDAP server.
2. Add the DN attributes for the CRL directory path to the LDAP directory and change the attribute settings to allow anonymous access to the CRL data.
3. Obtain the CRL from the CA.
4. Publish the CRL to the local LDAP server.
5. Define a CRL location in DCM.
6. Assign the CRL location to the CA certificate.
7. Change the DCM application definitions to perform CRL checking.
8. Update the CRLs regularly.

F.3 Configuring the LDAP server

The LDAP server configuration is performed by using the Operations Navigator. In this scenario we used the wizard to perform the basic setup.

1. Start the Operations Navigator. You need a user profile with *ALLOBJ and *IOSYSCFG special authorities.
2. Under the iSeries server, expand to **Network->Servers** and click **TCP/IP**.
3. From the list of servers, right-click **Directory** and select **Configure** from the context menu. If a directory server is already set up on the system, the Reconfigure option is shown. In this case, you may use the following steps as a reference and add the required suffix manually to not override an existing setup.
4. Select **Configure a local LDAP directory server** and click **Next**.

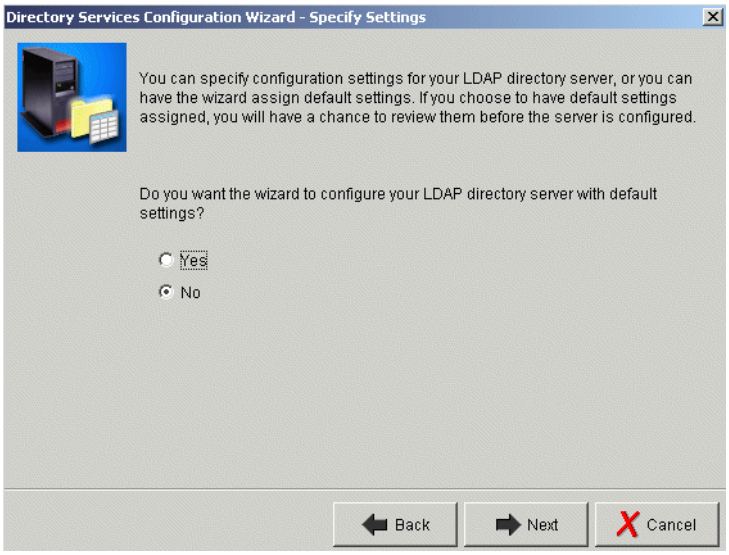


Figure 333. Directory Services Configuration Wizard - Specify Settings

5. Select **No** for the question shown in Figure 333 whether you want the wizard to create a configuration with default settings and click **Next**.

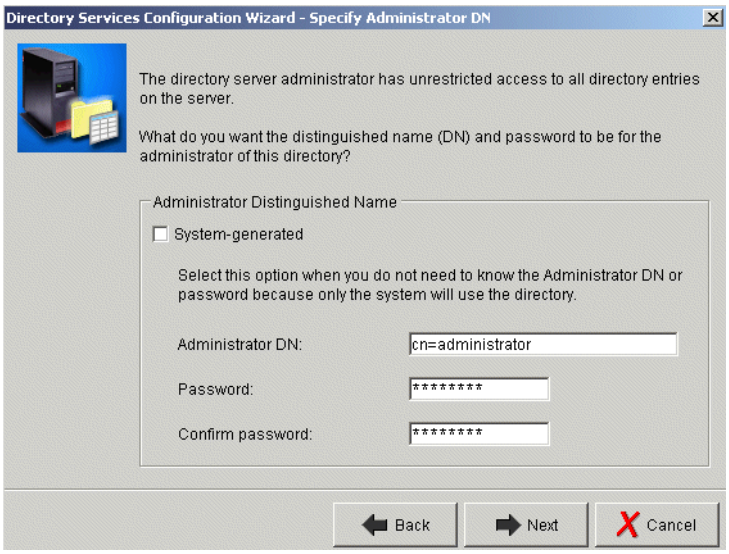


Figure 334. Directory Services Configuration Wizard - Specify Administration DN

6. Clear the **System-generated** box and enter a password for the administrator DN, as shown in Figure 334. Then click **Next**.

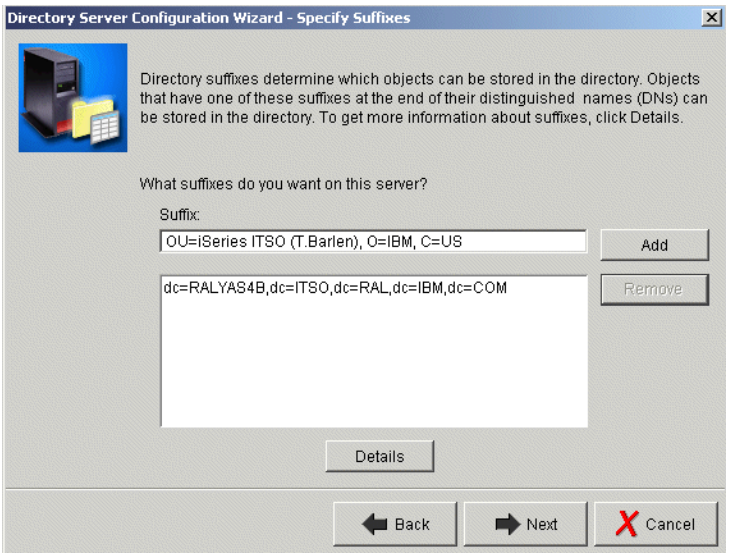


Figure 335. Directory Services Configuration Wizard - Specify Suffixes

7. Enter the suffix that will later contain the CRL. In the example shown in Figure 335, the certificates issued by the CA do not contain CRL

Distribution Points. That means that the default search path is the DN of the certificate's issuer, but without the Common Name (CN) attribute. As described in F.1, "Planning" on page 451, you can display a certificate issued by the CA or the CA certificate itself to find out what the issuer subject's DN is. In this configuration it is:

```
CN=ITSO Raleigh Root CA, OU=iSeries ITSO (T.Barlen), O=IBM, C=US
```

The suffix does not contain the CN attribute. So the value entered as a new suffix is:

```
OU=iSeries ITSO (T.Barlen), O=IBM, C=US
```

Note that the added suffix does not create any attributes within the LDAP directory. The suffixes specified on this page define only which objects can be stored in the directory itself.

8. Click **Add** to add the new suffix to the configuration and click **Next**.
9. Answer **Yes** to the question whether the Directory Services server should start when TCP/IP starts and click **Next**.

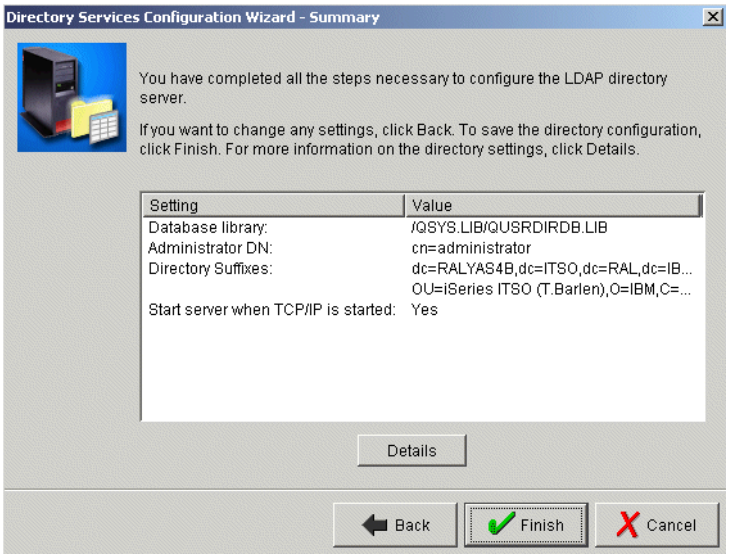


Figure 336. Directory Services Configuration Wizard - Summary

10. On the summary page shown in Figure 336, click **Finish** to complete the wizard.
11. The list of TCP/IP servers is displayed again. Right-click **Directory** and select **Start**.

F.4 Configuring the LDAP directory

Once the directory services (LDAP) server is configured, you need to add attributes to the directory. You also need to modify some attribute settings to allow anonymous access to the CRL. For this task we used the Directory Management Tool (DMT) provided with OS/400. You find the setup program for the DMT in the IFS directory /QIBM/ProdData/OS400/DirSrv/UserTools/Windows.

1. Start the Directory Management Tool (DMT). By default, you start the DMT in Windows by selecting the following path from the task bar:

   Start->Programs->IBM SecureWay Directory->Directory Management Tool

   If you receive a message that no connection could be established to the LDAP server, click **OK**. The DMT main window appears.

2. Click **Add server**.

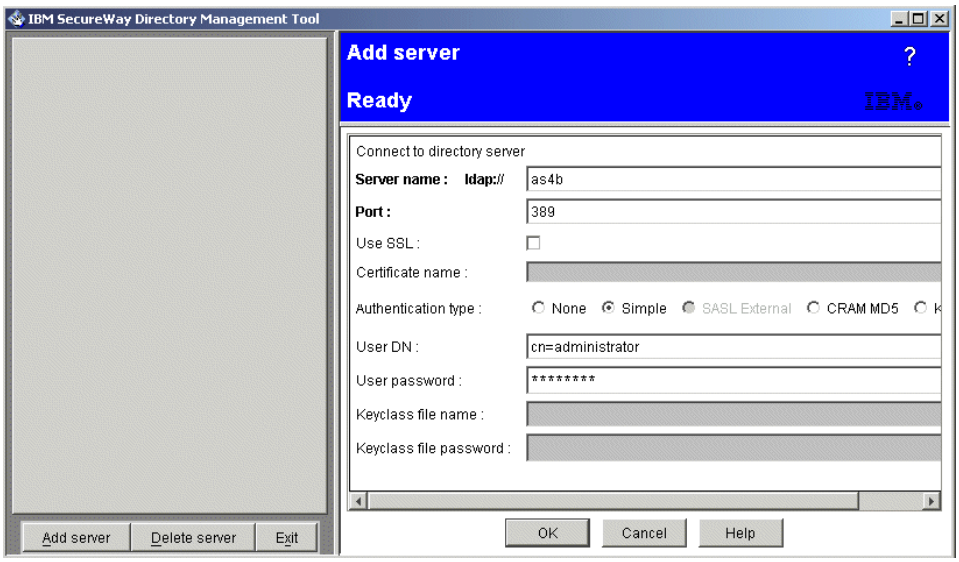


Figure 337. IBM SecureWay Directory Management Tool

3. In the window shown in Figure 337, enter the host name or IP address of your local LDAP server and the administrator DN and password as specified during the Directory Services Configuration Wizard. Click **OK.**

4. Under Directory tree click **Browse tree** in the DMT navigation pane. If you receive information messages that no data exists in one or more suffixes, click **OK** to close the message window.

5. Click **Add** on the right pane of the DMT window.
6. As shown in Figure 338, select the **Organizational unit** entry type and enter the suffix you entered in the Directory Service Configuration Wizard and click **OK**. This will create the attribute directory structure that will hold the CRL.

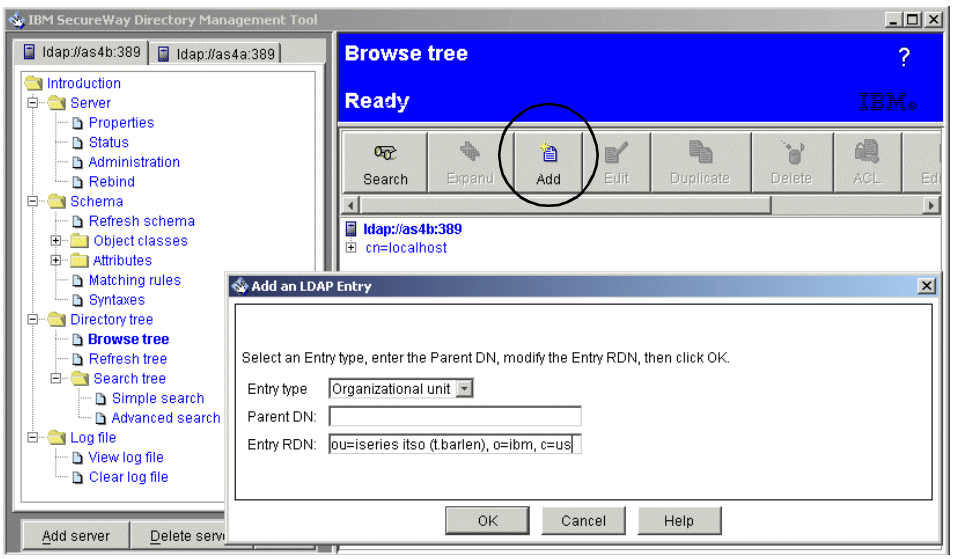


Figure 338. Adding an LDAP entry

7. Verify that the Object class is Organizational unit and that the data entered in the previous window is shown in the DN parameter. The ou: parameter should contain the DN without the “ou=” character string. Click **Add** to add the new entry.

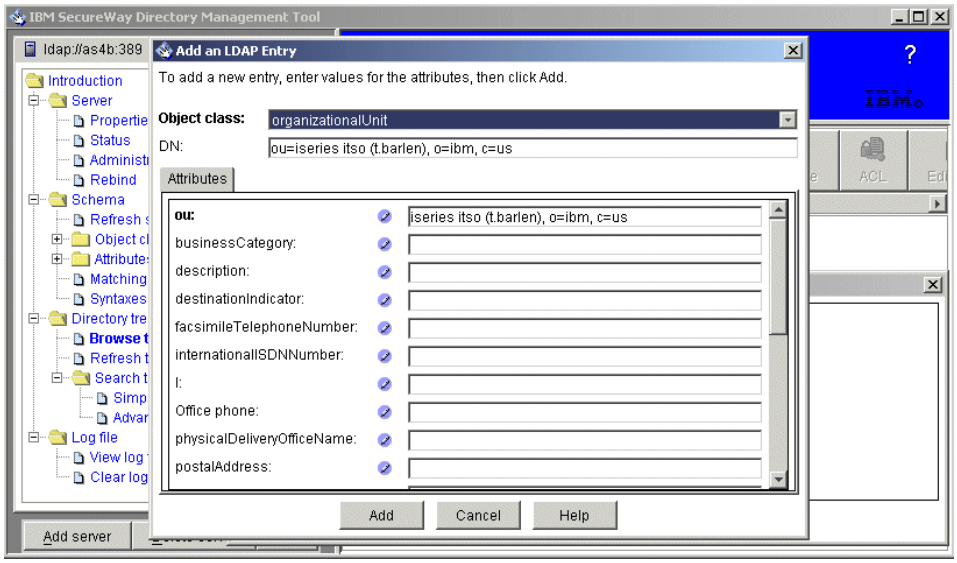


Figure 339. Adding an LDAP entry (2)

The next few steps are required to modify some attribute settings to allow anonymous access to the certificateRevocationList and authorityRevocationList attributes. By default, these attributes have their security class set to critical, which disallows anonymous access. An alternative to changing these attribute settings is to log in to the LDAP server with a user that has the appropriate authority.

8. From the DMT navigation pane expand **Introduction->Schema->Attributes** and click **Edit**.
9. From the Attribute name parameter in the Edit attribute window shown in Figure 339, select the **certificateRevocationList** attribute.

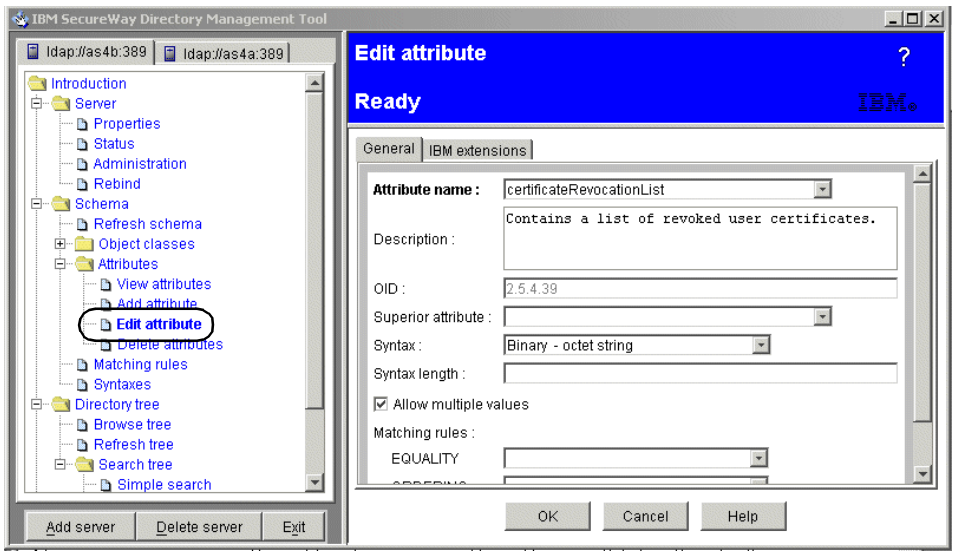


Figure 340. Edit attribute (1)

10. Click the **IBM extensions** tab. You will see Figure 341. Select **Normal** at the Security class parameter.

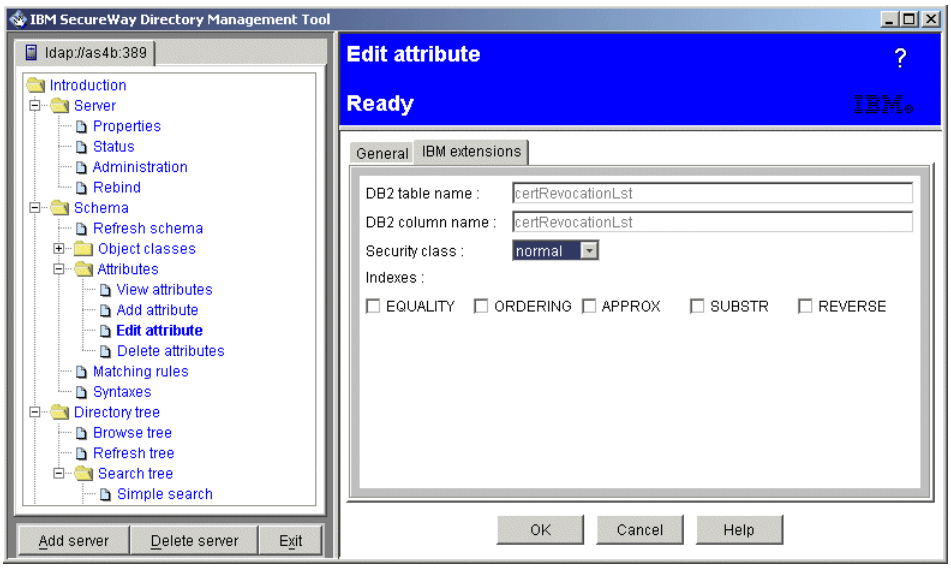


Figure 341. Edit attribute (2)

11. Click **OK** to save the changes.

12. For the authorityRevocationList attribute, also change the Security class to Normal by performing the previously described steps.
13. Close the DMT.

F.5 Obtaining the CRL from the CA

As mentioned at the beginning of this appendix, the methods for obtaining a CRL from a CA vary from CA to CA. You need to contact the CA for which you want to perform CRL checking on the local LDAP server to find out how to obtain the CRL.

In this example, we FTPed the CRL from the CA and stored it in the following directory path in the IFS:

/crl/IBMITSORoot.crl

You also need the CA certificate stored in a file for publishing the CRL to the LDAP directory. In this example, we obtained the CA certificate in binary format (DER encoded) and stored it into the crl directory.

/crl/W2KCA.CER

F.6 Publishing the CRL to the local LDAP directory (first time)

There are two ways of publishing a CRL to the local LDAP directory. You can use the DMT or use the command line interface in QShell. In this example we use the command line interface. To publish information into an LDAP directory using the command line interface, LDAP Directory Interchange Format (LDIF) files are used. The format consists of a version number, distinguished name, and attribute types and values.

The LDIF information described in this section is used to publish the CRL including the CRL's common name to the LDAP directory. The steps described in this section need to be performed only once per CA.

1. Sign on to the iSeries or AS/400 server and enter the following command:

   ```
   EDTF STMF('/crl/PublishCRL')
   ```

   Note that the crl directory existed and already contains the downloaded CRL.

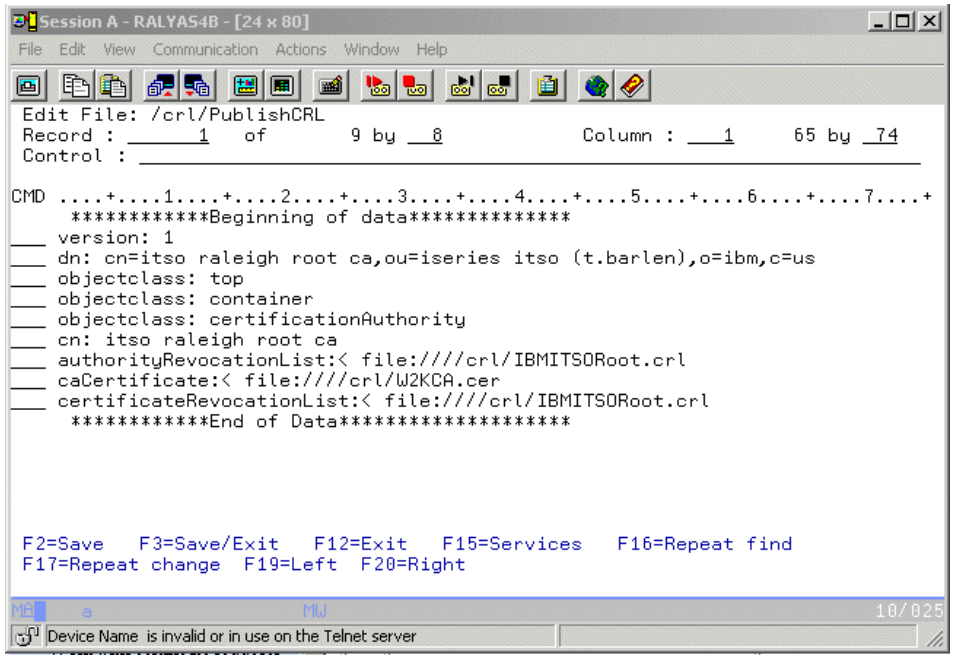


Figure 342. Entering LDIF information

Enter the data in the format shown in Figure 342. In the second line enter the full distinguished name of the CA's issuer subject. If CRL Distribution Points are used, it is the value provided by the CA. For the common name (cn) parameter enter the common name attribute of the CA's issuer subject or CRL Distribution Point. Do not add any suffixes or other attributes in this line.

The remaining three lines identify the CRL, CA certificate, and Authority Revocation List (ARL) to be published. ARLs have the same format as CRLs, but they contain revoked CA certificates. This is mostly used for intermediate CAs. All three attributes must be published to the LDAP directory as specified in the LDAP object and attribute definitions. In this example, the CRL is stored in /crl/IBMITSORoot.crl. The CA certificate is also stored in the /crl directory under the file name W2KCA.cer. Since the ARL must be published, but we do not use one, we decided to publish the CRL for the ARL also.

2. After you entered all required information, save it and exit the editor.
3. On the OS/400 command line, enter `QSH` to start the Qshell interface.

4. Enter the following command to publish the CRL into the LDAP directory:

   ```
   ldapadd -D "cn=administrator" -w "password" -f /crl/PublishCRL
   ```

   In this example, the `-D` parameter specifies the administrator DN as previously configured during the LDAP server setup in F.3, “Configuring the LDAP server” on page 452. The `-w` parameter specifies the administrator’s password that is used to log on to the LDAP server. And finally, the `-f` parameter specifies the path and file name of the LDIF file.

   You should get a confirmation message as shown in Figure 343.

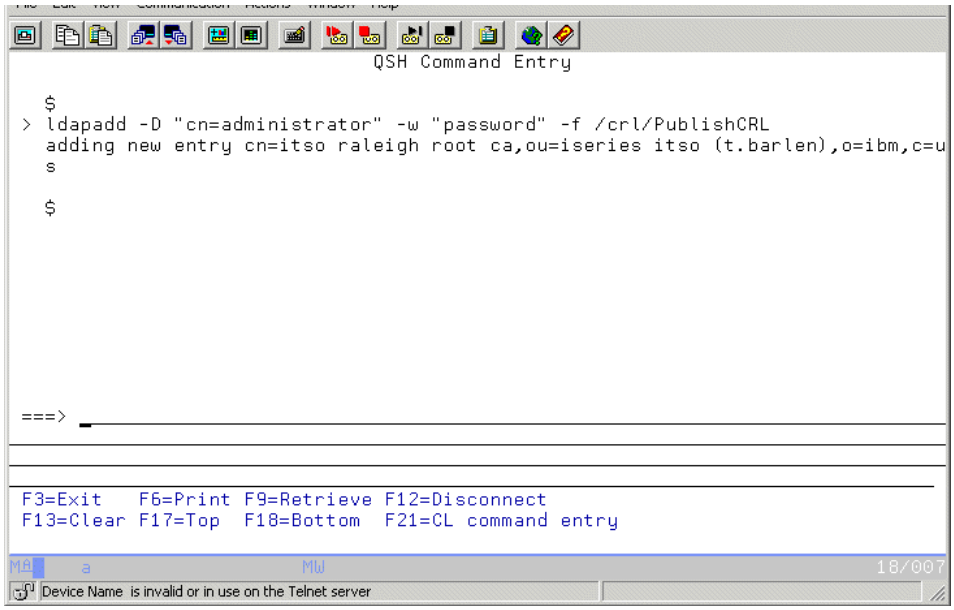


Figure 343. Publishing the CRL

F.7 Defining the CRL location in DCM

Now, that you have a working LDAP server with a published CRL, you can enable CRL checking for the CA. This process consists of three tasks. First, you need to add information about the location of the LDAP server. This is done by adding a CRL location using the Digital Certificate Manager (DCM) interface. It is assumed that the *SYSTEM certificate store is already configured.

1. Using a Web browser, start the AS/400 Tasks page by entering the following URL:

   ```
   http://as400hostname:2001
   ```

2. Click **Digital Certificate Manager**.
3. From the DCM navigation pane, click **Manage CRL Locations**. If you do not see the Manage CRL Locations link, you signed on with a user profile that does not have the required special authorities.
4. Under Manage CRL Locations, click **Add CRL location**.

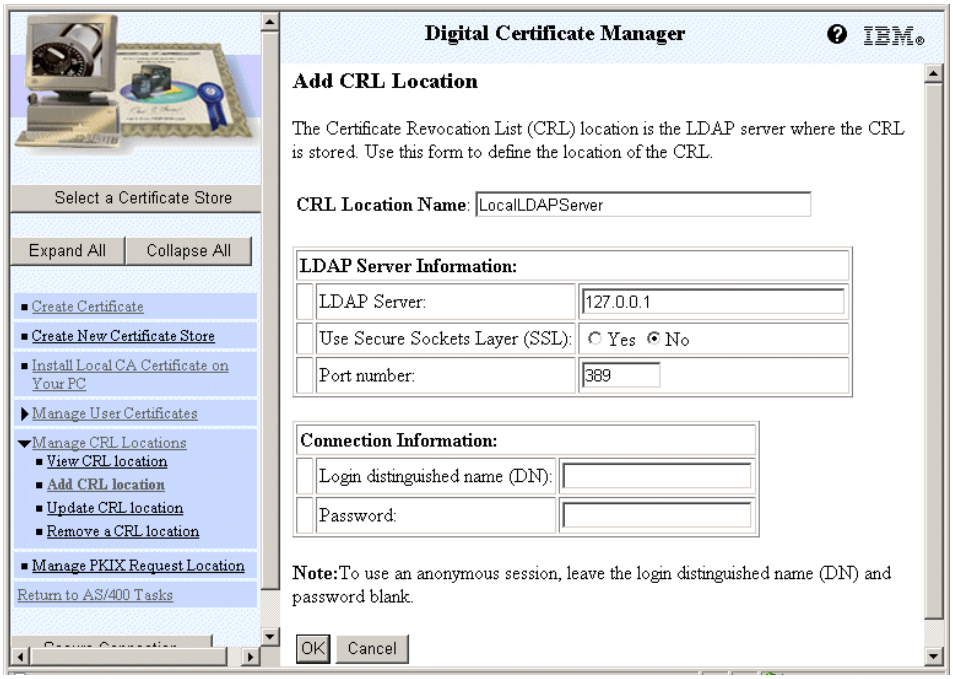


Figure 344. Add CRL Location window

5. Enter a descriptive name for the new CRL location. For the LDAP Server parameter, enter the loopback IP address of this iSeries or AS/400 server. Since the LDAP server resides on the same system and the loopback address is used, all LDAP traffic is processed internally in the system. Thus, SSL is not required. The default port for non-SSL LDAP connections is 389. For SSL connections, it is 636. Because the Security class attributes for the certificateRevocationList and authorityRevocationList attributes are set to Normal, anonymous login is sufficient. Therefore, the Login DN and password fields are blank.
6. Click **OK** to save the new CRL location.

F.8 Assigning the CRL location

This is the second task in enabling CRL checking for a particular CA. For more information about CRL locations and how to assign them to CA certificates, refer to 2.5, “Exploiting the Certificate Revocation List support” on page 82.

1. From the DCM navigation pane, click **Select a Certificate Store**.
2. Select the ***SYSTEM** certificate store and click **Continue**.
3. Enter the certificate store password and click **Continue**.
4. From the DCM navigation pane, click **Fast Path**.
5. Click **Work with CA certificates** from the Fast Path options.

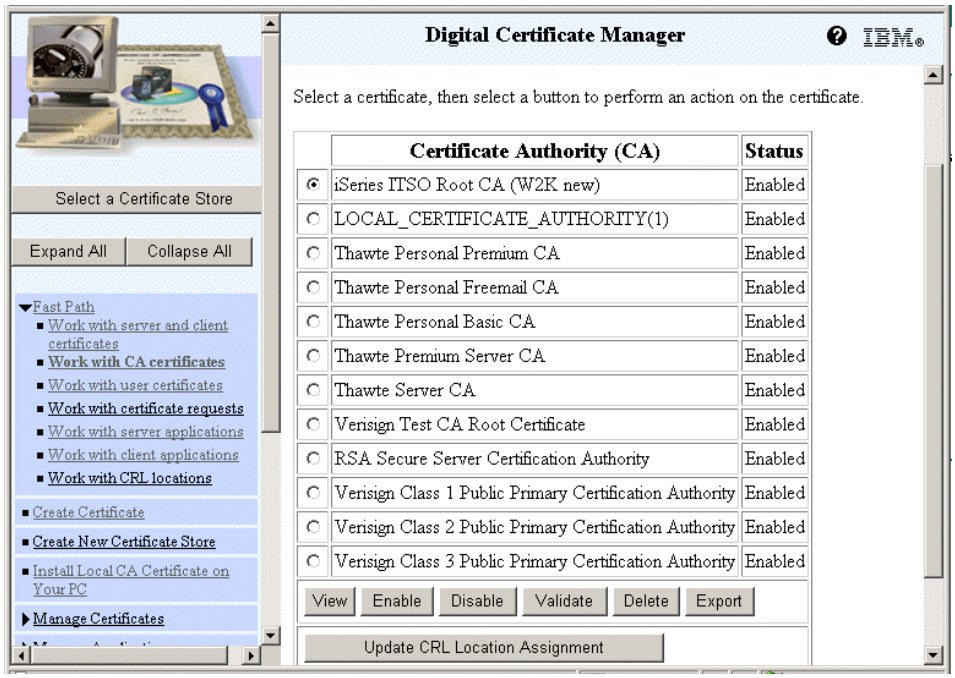


Figure 345. Work with CA Certificates window

6. Select the CA certificate you want to assign the CRL location for, as shown in Figure 345, and click **Update CRL Location Assignment**.

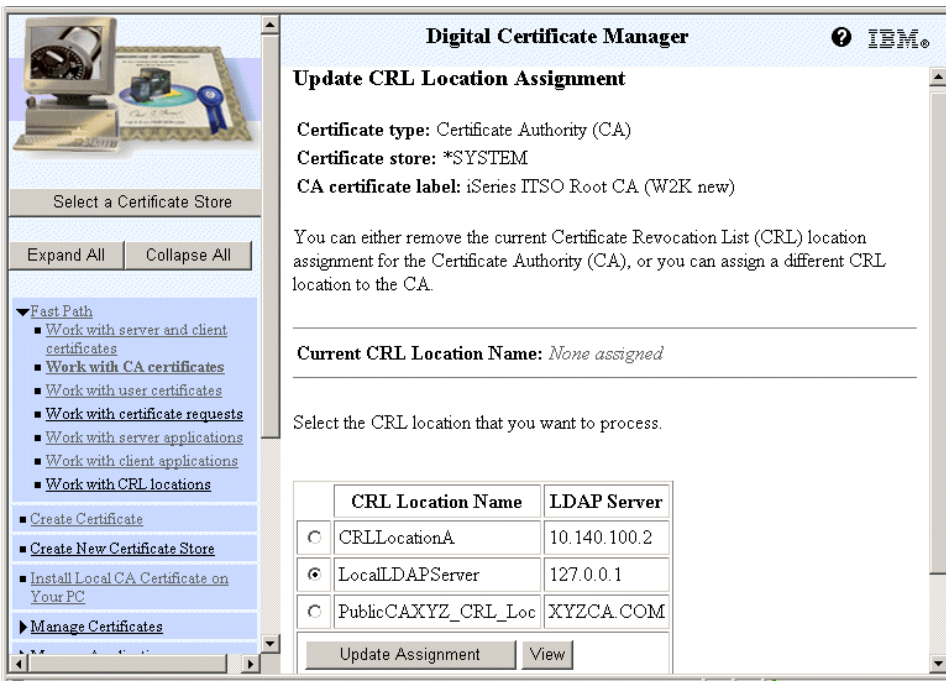


Figure 346. Update CRL Location Assignment

7. In the window shown in Figure 346, select the CRL Location Name you created in the previous task and click **Update Assignment**. The CRL location will be assigned to this CA certificate and you return to the Update CRL Location Assignment window.
8. Click **Cancel** to return to the list of CA certificates.

F.9 Enabling an application for CRL checking

The last task in setting up CRL checking is to enable it for a particular application. In this example, you see how to enable CRL checking for the OS/400 TCP/IP Telnet Server. However, the same method applies for all other applications, whether they are server, client, or object signing applications. The exception is VPN, which is also covered in this section.

F.9.1 Updating application definitions in DCM

The following steps enable CRL checking for applications managed by DCM. It is assumed that DCM is started and the *SYSTEM certificate store already opened.

1. From the DCM navigation pane, expand the **Fast Path** options.
2. From the Fast Path options, click **Work with server applications**.

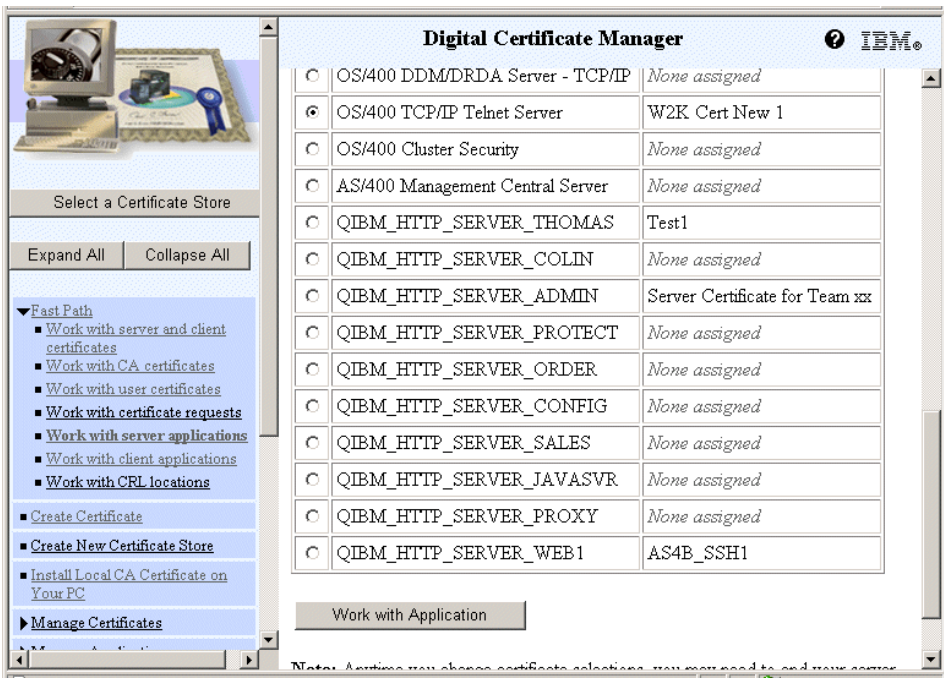


Figure 347. Work with server applications

3. In the window shown in Figure 347, select the application you want to enable CRL checking for and click **Work with Application**.

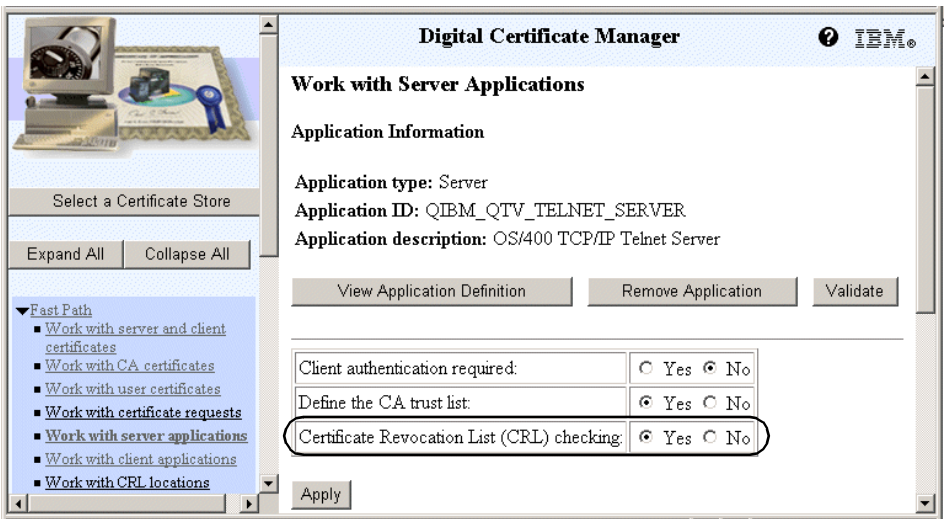


Figure 348. Work with Server Applications window

4. In the window shown in Figure 348, select **Yes** for **Certificate Revocation List (CRL) checking** and click **Apply**. The application information are redisplayed with a confirmation message.
5. Click **Cancel** to return to the list of server applications.

CRL checking is now active.

F.9.2 Enabling CRL checking for VPN connections

For VPN you also have to define first the CRL location and then assign the CRL location to the CA certificate. Then you need to change the VPN server settings as described in this section to enable CRL checking.

1. Start the Operations Navigator.
2. Expand the system to **Network->IP Policies** and right-click **Virtual Private Networking**. The window shown in Figure 349 is displayed.

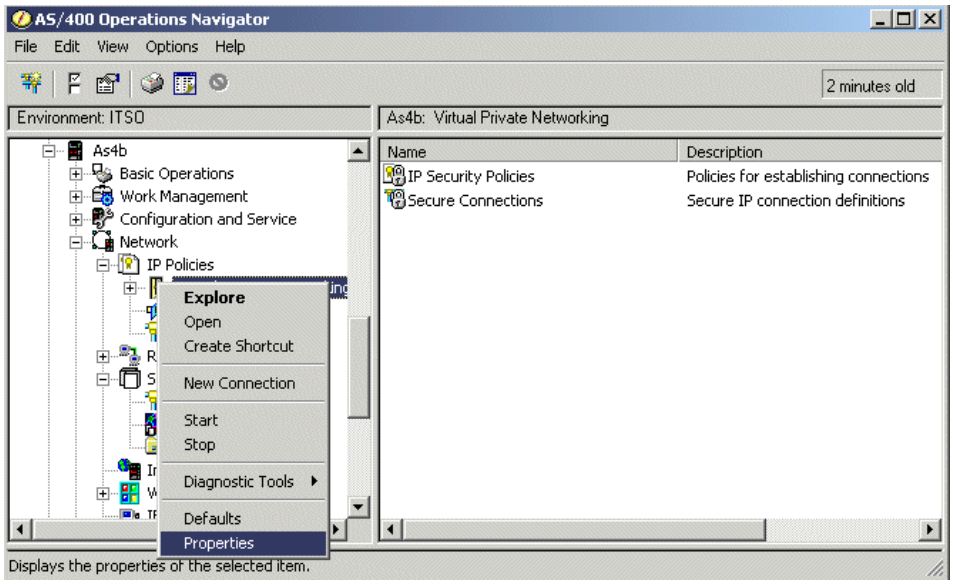


Figure 349. Operations Navigator

3. From the Virtual Private Networking context menu, select **Properties**. You see the display in Figure 350.

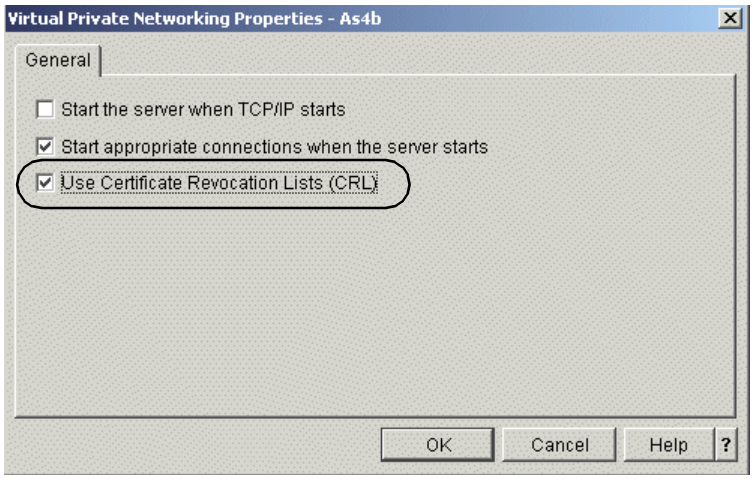


Figure 350. Virtual Private Networking Properties

4. Select **Use Certificate Revocation Lists (CRL)** and click **OK**.

F.10 Updating the CRL in the local LDAP directory

As mentioned in F.1, “Planning” on page 451, a CRL is valid only for a certain time. The CRL contains a time stamp when the next updated CRL will be published. Since you cannot easily retrieve this information, you have to contact the CA to find out in which interval the CA publishes the CRLs. Note that for systems that support CRL caching, the time stamp for the next update is automatically retrieved. Once you know the schedule, you can write a simple CL program that will automatically update the CRL in the local LDAP directory. The steps shown in this section show you how to define an LDIF file for updating a CRL and the command to actually perform the update.

1. Sign on to the iSeries or AS/400 server and enter the following command:

   ```
   EDTF STMF('/crl/UpdateCRL')
   ```

   This command opens a file UpdateCRL that will contain the necessary LDIF information for updating the CRL on the local LDAP server.

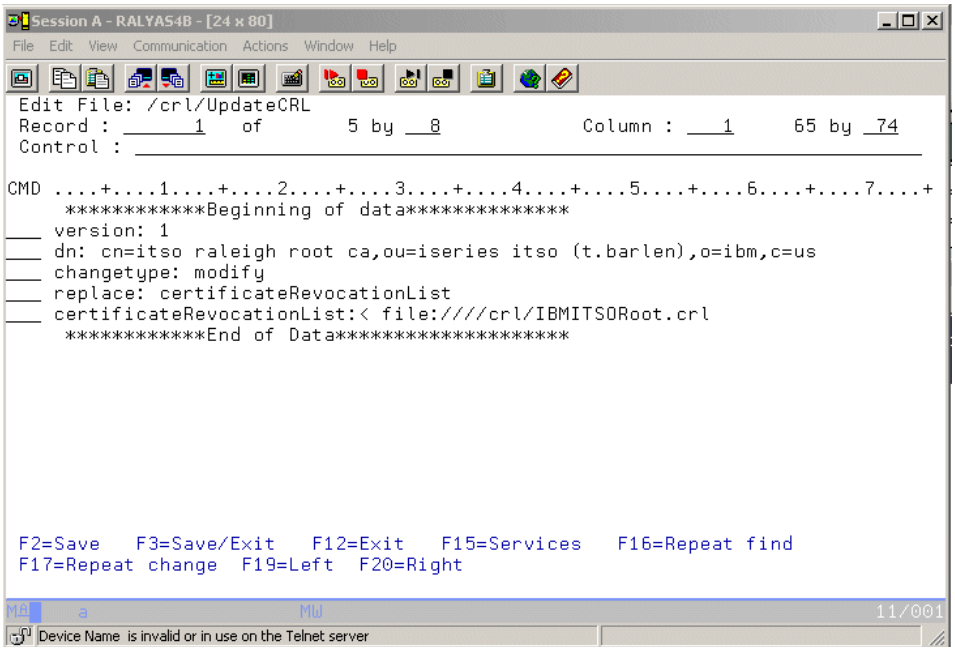


Figure 351. Adding LDIF information for the CRL update

Enter the information as shown in Figure 351. The first two lines of the file are the same as described in F.6, “Publishing the CRL to the local LDAP directory (first time)” on page 460. Then you have to enter the changetype, which is in this case “modify”. The next line contains information about

what attribute you want to replace. In this case you update only the certificateRevocationList attribute. Note that when you also use ARLs or the CA certificate has changed, you may want to update these attributes as well. In the last line you specify the path and file name of the new CRL.

2. Once all the information is entered, save it and exit the editor.
3. You can update the CRL by calling the update program manually or by creating a simple CL program that can be automatically started using the OS/400 job scheduler. The following command needs to be performed manually or added to the CL program to update the CRL in the local LDAP directory. Remember, you need to obtain the updated CRL first.

```
QSH CMD('ldapmodify -D "cn=administrator" -w "password" -b -r -f
/crl/UpdateCRL')
```

Running the `ldapmodify` command manually via the Qshell interface is shown in Figure 352.

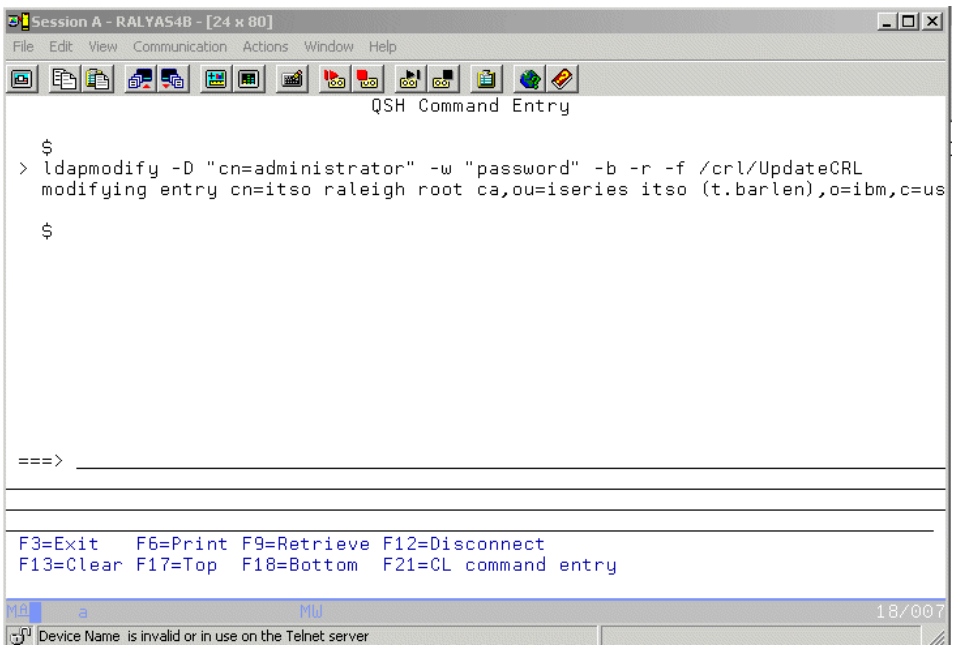


Figure 352. Updating the CRL via Qshell

Appendix G. Using the additional material

This redbook also contains additional Web material. See the appropriate section below for instructions on using or downloading each type of material.

G.1 Locating the additional material on the Internet

The CD-ROM, diskette, or Web material associated with this redbook is also available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

ftp://www.redbooks.ibm.com/redbooks/SG246168

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number.

G.2 Using the Web material

The additional Web material that accompanies this redbook includes the following:

File name	*Description*
code.zip	Zipped Code Samples
readme.html	Information about how to restore and use the downloaded material

G.2.1 System requirements for downloading the Web material

The following system configuration is recommended for downloading the additional Web material.

Hard disk space:	5 MB minimum
Operating System:	Windows
Processor:	486 or higher
Memory:	Minimum requirements for Windows operating system

G.2.2 How to use the Web material

Create a subdirectory (folder) on your workstation and copy the contents of the Web material into this folder. Then follow the instructions provided in the README.HTML file.

Appendix H. Special notices

This publication is intended to help network administrators and programmers who are in charge of planning and implementing network and system security to understand and use the new security enhancements as introduced with OS/400 Version 5 Release 1. The information in this publication is not intended as the specification of any programming interfaces that are provided by IBM Operating System/400. See the PUBLICATIONS section of the IBM Programming Announcement for OS/400 V5R1 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee

that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

AIX	Netfinity
APPN	Operating System/400
AS/400	OS/400
AS/400e	RACF
AT	Redbooks
e (logo)®	Redbooks Logo
DRDA	RPG/400
IBM ®	S/390
Integrated Language Environment	SecureWay
Language Environment	400

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere.,The Power To Manage., Anything. Anywhere.,TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States and/or other countries.

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and other countries licensed exclusively through The Open Group.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Appendix I. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

I.1 IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 479.

- *AS/400 Internet Security: Developing a Digital Certificate Infrastructure,* SG24-5659
- *AS/400 Internet Security Scenarios: A Practical Approach,* SG24-5954
- *iSeries and AS/400e System Builder Version 4 Release 5,* SG24-2155
- *AS/400e System Handbook,* GA19-5486
- *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149

I.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at `ibm.com/redbooks` for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
IBM System/390 Redbooks Collection	SK2T-2177
IBM Networking Redbooks Collection	SK2T-6022
IBM Transaction Processing and Data Management Redbooks Collection	SK2T-8038
IBM Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
IBM AS/400 Redbooks Collection	SK2T-2849
IBM Netfinity Hardware and Software Redbooks Collection	SK2T-8046
IBM RS/6000 Redbooks Collection	SK2T-8043
IBM Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

I.3 Referenced Web sites

These Web sites are also relevant as further information sources:

- `http://www.rsasecurity.com` Information about RSA encryption and authentication algortihms
- `http://www.ibm.com/security/cryptocards/html/library.shtml` 4758 Cryptographic Coprocessor product information and documention
- `http://www.as400.ibm.com/tstudio/tech_ref/security/crypto/index.htm` PCI Cryptographic Coprocessor information in iSeries Technical Studio
- `http://www.seagullsw.com` Product information about the BlueZone Secure FTP client
- `http://digitalid.verisign.com/server` Requesting a server GlobalID certificate from VeriSign
- `http://crl.verisign.com` Download site for VeriSign's Certificate Revocation Lists (CRLs)
- `http://www.as400.ibm.com/tstudio/tech_ref/security/crypto/index.htm` 4758 PCI Cryptographic Coprocessor for iSeries information
- `http://www.fortify.net/sslcheck.html` Checking SSL capabilities of your Web browser
- `http://www.tml.hut.fi/Studies/Tik-110.350/1998/Essays/ssl.html` Introduction and overview of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
- `http://support.microsoft.com/support/kb/articles/Q216/4/82.ASP` Support document describing how to control ciphers with Microsoft's Internet Explorer

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** ibm.com/redbooks

  Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

  Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

  Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the “How to Order” section at this site: http://www.elink.ibmlink.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at http://w3.itso.ibm.com/ and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at http://w3.ibm.com/ for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

☐ Invoice to customer number

☐ Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Abbreviations and acronyms

AES	Advanced Encryption Standard
API	Application Programming Interface
ARL	Authority Revocation List
ASN	Abstract Syntax Notation
BER	Basic Encoding Rules for ASN.1
CA	Certificate Authority
CCSID	Coded Character Set Identifier
CRL	Certificate Revocation List
CRMF	Certificate Request Message Format
CSR	Certificate Signing Request
DCM	Digital Certificate Manager
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DNS	Domain Name Services
FIPS	Federal Information Processing Standard
FQDN	Fully Qualified Domain Name
GSKit	Global Secure ToolKit
HTML	Hypertext Markup Language
HTTP	Hypertext Transport Protocol
IBM	International Business Machines Corporation
IETF	Internet Engineering Task Force
ILE	Integrated Language Environment
IP	Internet Protocol
IPSec	Internet Protocol Security framework
ITSO	International Technical Support Organization
LDAP	Lightweight Directory Access Protocol
MAC	Message Authentication Header
OPM	Original Program Model
OS/400	IBM Operating System/400
PKCS	Public-Key Cryptography Standards
PKA	Public Key Algorithm
PKI	Public Key Infrastructure
RFC	Request for Comments
SHA	Secure Hash Algorithm
SSL	Secure Socket Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
URL	Universal Resource Locator
VPN	Virtual Private Networking

Index

Symbols

Numerics

A

B

C

D

E

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at ibm.com/redbooks
- Fax this form to: USA International Access Code + 1 845 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number **Redbook Title**	SG24-6168-00 IBM @server iSeries Wired Network Security: OS/400 V5R1 DCM and Cryptography Enhancements
Review	
What other subjects would you like to see IBM Redbooks address?	
Please rate your overall satisfaction:	O Very Good O Good O Average O Poor
Please identify yourself as belonging to one of the following groups:	O Customer O Business Partner O Solution Developer O IBM, Lotus or Tivoli Employee O None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	O Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. ibm.com/privacy/yourprivacy/

IBM

Redbooks

IBM eserver iSeries Wired Network Security

(1.0” spine)
0.875”<->1.498”
460 <-> 788 pages

IBM @server iSeries Wired Network Security

OS/400 V5R1 DCM and Cryptography Enhancements

Improve SSL performance with the 4758 Cryptographic Coprocessor

SSL Sockets application examples using the GSKit APIs

Ensure object integrity with object signing

With the increasing number of customers that conduct business over the Internet or other untrusted networks, there is a rising demand to protect data traffic. This IBM Redbook focuses on the network security enhancements that are introduced with OS/400 Version 5 Release 1. You learn how to implement and use the new object signing capabilities, so Business Partners and customers can distribute objects over an untrusted network while assuring their integrity. You are guided through the redesigned Digital Certificate Manager (DCM) with its new functions, such as Certificate Revocation List processing.

For the e-commerce world, availability, security, and performance are critical to business. This redbook introduces the new 4758 Cryptographic Coprocessor support, which helps improve SSL performance and security. It takes you through the cryptographic coprocessor configuration and explains how to use it by DCM.

This redbook introduces the new Global Secure Toolkit (GSKit) APIs that provide better functions and more flexibility when writing SSL Sockets applications. You'll find sample code written in ILE RPG to introduce these new APIs.

This is the first publication to provide complete information about the supported encryption and authentication algorithms and key lengths. It shows how to control your Web server to accept certain ciphers for a secure connection using the new SSL directives.

SG24-6168-00 ISBN 0738422169

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks