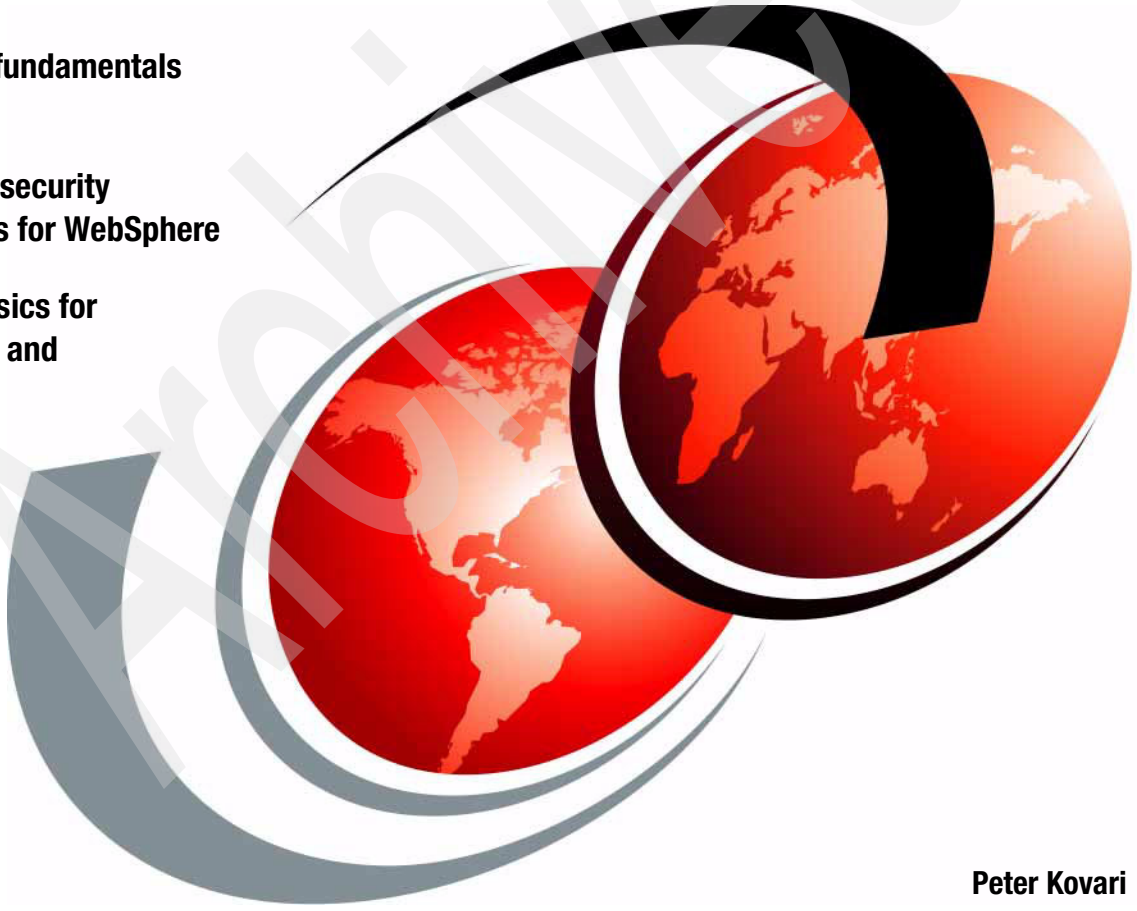
WebSphere® software

IBM

WebSphere Security Fundamentals

- IT security fundamentals
- Supporting security components for WebSphere
- Security basics for J2SE, J2EE, and WebSphere



Peter Kovari

Redpaper

ibm.com/redbooks

International Technical Support Organization

WebSphere Security Fundamentals

April 2005

Note: Before using this information and the product it supports, read the information in “Notices” on page v.

First Edition (April 2005)

This edition applies to WebSphere Application Server V4, V5, V6

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described 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:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

Domino®	Redbooks™	WebSphere®
IBM®	Redbooks (logo)  ™	

The following terms are trademarks of other companies:

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redpaper is an overview of IT security for WebSphere® products. Its purpose is to provide basic general information about IT security and specific details about WebSphere security. This can be used as a standalone paper about basic security, or as an unofficial introduction to WebSphere V6 security and future security-related Redbooks™.

Chapter 1, “Security fundamentals” discusses basic IT security without reference to any product.

Chapter 2, “Supporting security components for WebSphere” starts introducing IT components related to security without specifying product names.

Chapter 3, “Security fundamentals for J2SE, J2EE and WebSphere” goes into detail about J2EE and IBM WebSphere security. It provides an overview of security-related technologies and functions.

If you are not familiar with IT security in the Java™ 2 and WebSphere environments, this paper should be a good start.

The team that wrote this Redpaper

This Redpaper was produced at the International Technical Support Organization, Raleigh Center.

Peter Kovari is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies, and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us because we want our papers to be as helpful as possible. Send us your comments about this Redpaper or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HZ8 Building 662
  P.O. Box 12195
  Research Triangle Park, NC 27709-2195

1

Security fundamentals

This chapter introduces some of the security fundamentals that are related to e-business applications in the e-business world.

This document is not intended to cover every aspect of security, or even every aspect of e-business security. Rather, it is a short overview of security topics related to WebSphere Application Server.

1.1 Security

As new business practices emerge, most enterprises are finding that their existing security infrastructure is not capable of meeting the rapidly changing and more rigorous demands of business over the Internet. The demands of network security have now gone far beyond simply managing user accounts and restricting access between internal and external networks. These demands now require a sophisticated system with fine-grained access control to resources, yet that is manageable enough to be tailored to protect systems from many types of security threats.

Security is a vast topic; everything involves security to some extent, in a certain format. There are two main areas that have to be discussed separately:

- Physical security
- Logical security

Systems must be protected both from outsiders and insiders. Not every intrusion or attack is intentional; misuse of a system or improper administration can also cause damage.

1.1.1 Physical security

Physical security means protection against physical actions. It involves every physical element around:

- Any machine where the application is running
- The room where the machines are operating
- The building where the machines are installed
- The site where the company is located

The listed elements must be secured against intrusion and damage, whether it be intentional or not.

Physical security also includes the protection of communication channels:

- Ground lines
- Wireless connections

The communication network must be protected against eavesdropping and damage to the connection (such as cutting the line).

The subject of physical security extends far beyond the objective of this paper. This short section is intended only as a reminder of the concept of physical security.

1.1.2 Logical security

Logical security is related to particular IT solutions: the IT architecture and applications, including the business processes.

Communication

Network communication must be protected not only on a physical level but on a logical level as well. Most companies' networks are connected to public networks, making applications accessible from the outside world. Network-level security must prevent unauthorized access.

Application

An application is secured on different levels. Security is involved from the very beginning of the implementation, when the processes and flows are designed.

- Securing the resources

  This implies protecting the resources on an application level and exercising the security features of the runtime platform (authentication and authorization).

- Implementing the business processes securely

  The processes must be designed in a way that no weakness in logic can be found.

1.1.3 Security policy

Security policies are guidelines for an organization; they can be part of a widely accepted standard (ISO) or implemented by a certain organization or company.

Policies can define processes for different areas in an organization. Security policies focus on security-related processes (for example, how to request a new password, how to renew a password, and so on).

These guidelines are very important in implementing a robust security policy for the whole system organization-wide.

1.2 Security fundamentals

This section discusses two fundamental security services that are also supported by WebSphere Application Server:

- Authentication
- Authorization

1.2.1 Authentication

Authentication is the process of establishing whether a client is valid in a particular context. A client can be an end user, a machine, or an application.

Definition: A *realm* is a collection of users that are controlled by the same authentication policy.

The authentication process involves gathering some unique information from the client. Three major groups of secure authentication are used to gather this unique information:

- Knowledge-based: user name and password, for example
- Key-based: physical keys, encryption keys, and key cards
- Biometric: fingerprints, voice patterns, and DNA

Other authentication mechanisms can combine these; an example is digital certificates, in which key-based and knowledge-based authentication are exercised.

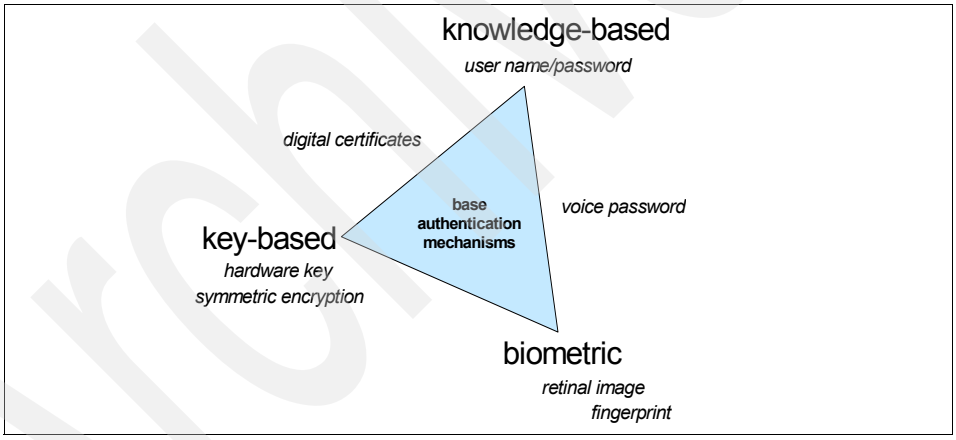


Figure 1-1 Base authentication mechanisms

The following paragraphs discuss some of the authentication mechanisms that are used in IT systems.

User name and password

User name and password are the most common method for authentication. The user who wants to access the system logs on with a user name and a password, which are compared to the values stored in the system.

Physical keys

Physical keys are objects that can be used to prove the identity of the object holder: the metal key that is used to unlock your computer, a hardware device that is plugged into the computer to execute certain programs, or a smart card with an embedded memory or microprocessor.

Biometric authentication

Biometric authentication is the use of physiological or behavioral characteristics to verify the identity of an individual. Biometric authentication consists of comparing the physical characteristics of an individual against the values of those characteristics stored in a system.

Re-authentication

Re-authentication occurs when the user is required to authenticate again some time after a successful authentication. Re-authentication might be used in any of these cases:

- The user is trying to access a new application that does not belong to the security domain where the user is already logged in, and there is no single sign-on solution in place to avoid re-authentication.
- The time period for the user session expires after a certain time, so the user has to re-authenticate to continue the existing session or to start a new one.
- The user needs to change identity or add a new identity (subject) to the existing session.

Impersonation

Impersonation is when a subject (user or system) acts in behalf of another subject. An existing identity can pick up an additional, new identity or switch to another identity and use the new identity to present itself.

Delegation

Delegation is the ability to leave an intermediary to do the work initiated by a client according to a delegation policy.

For example, in a distributed object environment, a client can request the method of an object on server A. The method request results in invoking another method of an object in server B. Server A performs the authentication of the identity of the client and passes the request to server B. Server B assumes that the client identity has been verified by server A and responds to that request as shown in Figure 1-2 on page 6.

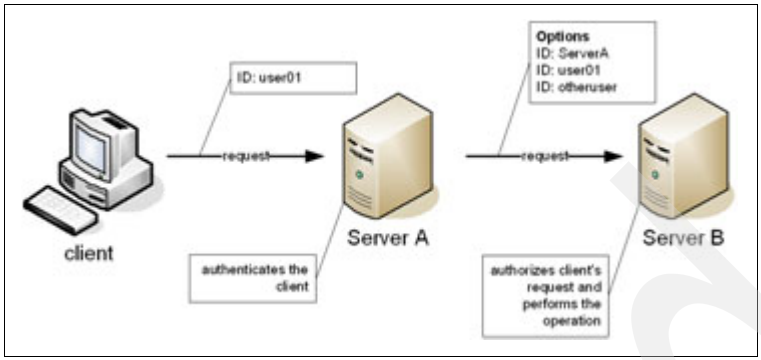


Figure 1-2 Delegation mechanism

Depending on the application environment, the intermediary can have one of the following identities when making a request to another server:

Client identity The identity under which the client is making the request to the intermediary

System identity The identity of the intermediary server

Specified identity The identity specified through configuration

Single Sign-On

Single Sign-On is a convenient function for environments with multiple secured applications. Most of these applications usually require authentication, but they:

- May have different authentication mechanisms
- May have different user registries
- May require a different identity from the same user

To avoid another authentication during the same session, Single Sign-On takes care of the further authentication steps on behalf of the user.

Single Sign-On can be a function supported on the application level or application server level, or it can be an additional component in the security infrastructure that takes care of authentication.

1.2.2 Authorization

Authorization is the process of checking whether the authenticated user has access to the requested resource. There are two fundamental methods:

- Access control list
- Capability list

Access control list

Each resource has associated with it a list of users and what each can do with the resource (for example: use, read, write, execute, delete, or create).

Usually, an access control list specifies a set of roles that are allowed to use a particular resource. It also designates the people allowed to act in these roles.

For example, in a bank account object, we can have different methods (transfer, deposit, getBalance, setInterest). The access right can be granted on the basis of the roles of the users within the organization. A bank teller can have access to the *getBalance* method but not to *setBalance,* while a manager can have access to both methods.

Table 1-1 Example of a Role Access Control List

Resources	Bank teller role	Manager role
getBalance method	yes	yes
setBalance method	no	yes

Capability list

Associated with each user is a list of resources and the corresponding privileges held for the user.

In this case, the holder is given the right to perform the operation on a particular resource.

In the previous example of the bank account object, the access right is granted to the user if the resource is listed in the user's capability list.

Table 1-2 Example of a capability list

Roles	getBalance method	setBalance method
Bank teller role	yes	no
Manager role	yes	yes

Table 1-1 and Table 1-2 are very similar, but the rows and columns are switched. Actually, this is the difference between the two approaches. We have two sets: roles and resources. In the first case, roles are mapped to resources, and in the second case resources are mapped to roles.

The access control list is exercised generally, because managing security for certain resources is easier and more flexible than mapping resources to roles.

Role-based security

Roles are different levels of security that relate to a specific application. Different employees have different roles, so the security access that each employee requires to complete the tasks in a Web application are also different. In a role-based authorization model, the roles for a given application are developed as the application is developed. As a user base for the application is established, one of three things happens:

- Users are mapped directly to specific security roles.
- Groups are formed, users are defined as members of a group, and the groups are defined to specific security roles.
- A combination of user/group mapping to security roles is used to handle any exceptions.

1.2.3 Secure communication

Secure communication protects transmitted data against eavesdropping. In a secured communication, the data is encrypted during the transport.

Note: There is a significant difference between encoding and encrypting.

Encoding is the process of transforming data into another form of representation based on rules. These rules define the mapping between the original character and their representation. The transformation is easily reversible.

Encrypting is the process of transforming data into an entirely new set of data. The transformation is based on complex algorithms to ensure that there is no (or very little) correlation between the original data and the transformed data. In an ideal situation, transformation is non-reversible; in reality it is very hard to reverse.

The secured communication always starts with a “handshake” or contract. During the handshake, the parties can agree on the nature and the details of the secure communication, for example: authenticate each other and agree on a method to use for encrypting the messages.

1.3 Security in use

Security is a complex and diversified topic, so it is important to keep it simple. Basic security areas include:

- Authentication / identification

  Measures designed to protect against fraudulent transmission and imitative communications by establishing the validity of transmission, message, station, or individual.

- Access control

  The prevention of improper use of a resource, including the use of a resource in an unauthorized manner.

- Privacy / confidentiality

  Assurance that information is not made available or disclosed to unauthorized individuals, entities, or processes.

- Data integrity

  The correctness of information, of the origin of the information, and of the functioning of the system that processes it.

- Accountability / non-repudiation

  Assurance that the actions of an entity may be traced uniquely to the entity. This ensures that there is information to prove ownership of the transaction.

- Administration / configuration

  Methods by which security policies are incorporated into the architecture and the functionality that the system architecture needs to support.

- Assurance / monitoring

  Confidence that an entity meets its security objectives; this is usually provided through an intrusion detection system.

- Security management

  Assurance that an entity meets its security management objectives, processes, and procedures.

If you keep this list in mind during design and development, security will be well implemented.

2

Supporting security components for WebSphere

This chapter is an introduction to the external (to the application server) components that support WebSphere to improve security for the whole environment. End-to-end security always includes more than the application server with the applications. Various other components work with the application server to provide security services for applications.

Some of the supporting, external components are discussed in this chapter.

2.1 User registry (or directory)

User registry is a key component for applications. It is responsible for holding information about users, identities, and groups for security domains. There are many different type of user registries available, including:

- LDAP (Lightweight Directory Access Protocol) server
- Database-based user registry
- Operating system's user registry
- File-based user registry

The most common user registry today is the LDAP directory, or other user registries supporting LDAP to query users (for example: Domino® directory).

User registries have different methods for organizing users in the registry. Generally users and groups are mapped to some type of hierarchy in the user registry, possibly following the company's organizational structure.

2.2 Authorization and authentication server

The authorization and authentication server (security server) is a key component in centralizing security in the company. The server can evaluate authentication and authorization requests on behalf of the application servers. Application servers have to externalize these security functions. In a proper environment the application servers trust the security server and they do not even do user registry lookups any more, leaving that to the security server.

Authentication is a relatively simple function to externalize. Most of the authentication mechanisms are well defined for the different application servers. Authentication is performed only a few times during a session, so externalizing such a function should not have a significant impact on performance.

Authorization is a more complex function in the sense that different application servers define authorization entries in different ways, and authorization can be applied to many different components and actions defined on the components. Centralizing such a function requires flexibility from the security server. An authorization check is performed several times during a session, every time a secured resource is accessed, so externalizing the function may make a significant performance impact.

2.3 Security reverse proxy server

The security reverse proxy server resides between the Web clients and the Web application server. This server's responsibility is to intercept incoming calls from the clients and perform authentication and authorization on behalf of the application server running the Web application.

By acting as a single point of entry for the clients, this server increases security in the environment. However, it can also be considered a bottleneck in the network infrastructure. Security reverse proxy servers are usually high-performance servers; or you can load-balance between a few of them for better performance.

2.4 Public Key Infrastructure (PKI)

PKI is closely related to cryptography. Although it seems complicated, it is not. We do not explain the details or go into low-level mathematical algorithms here, but you should understand the background involved.

Secret key cryptography

Secret key algorithms, which use one key to encrypt and decrypt the data, were invented before public key algorithms.

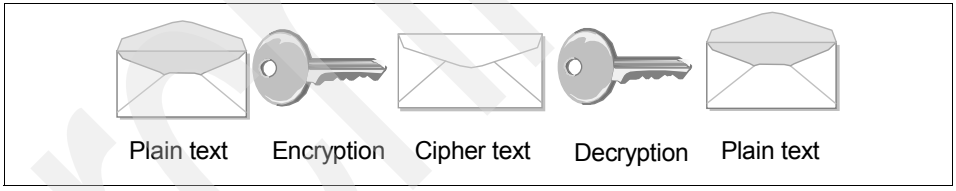


Figure 2-1 Symmetric key encryption

Figure 2-1 illustrates the concept of symmetric key cryptography. The algorithms that are used provide a great advantage: They are faster than public key cryptography. They have a considerable disadvantage as well: The same key is needed for encryption and decryption, and both parties must have the same keys. In today's cryptography, the secret keys do not belong to persons but to communication sessions. At the beginning of a session, one of the parties creates a session key and delivers it to the other party; they can then communicate securely. At the end of the session, both parties delete the key and, if they want to communicate again, must create another key.

The following section discusses how to secure the delivery of the session key.

Public key cryptography

The first imperative of public key cryptography is the ability to deliver session keys securely. It has many more benefits than secret key cryptography, as we show in the following section.

Public key cryptography involves the use of different keys for encrypting and decrypting functions. If you encrypt something with key 1, you can only decrypt it with key 2, as shown in Figure 2-2.

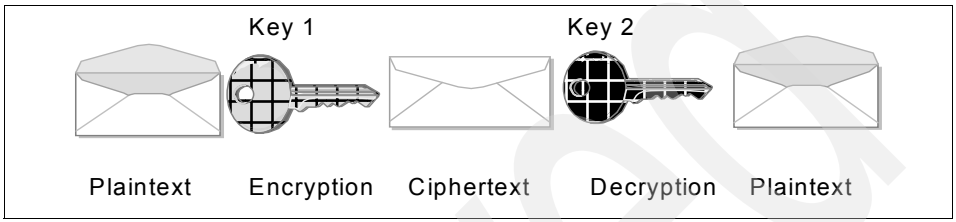


Figure 2-2 Public key concept

This architecture allows the use of one of the keys as a private key. This means that nobody can have access to this key except the owner. The other key can be used as a public key. If a user wants to send an encrypted message to another person, he or she will get the other person's public certificate, encrypt the message and send it. The message can be decrypted only by the owner of the private key.

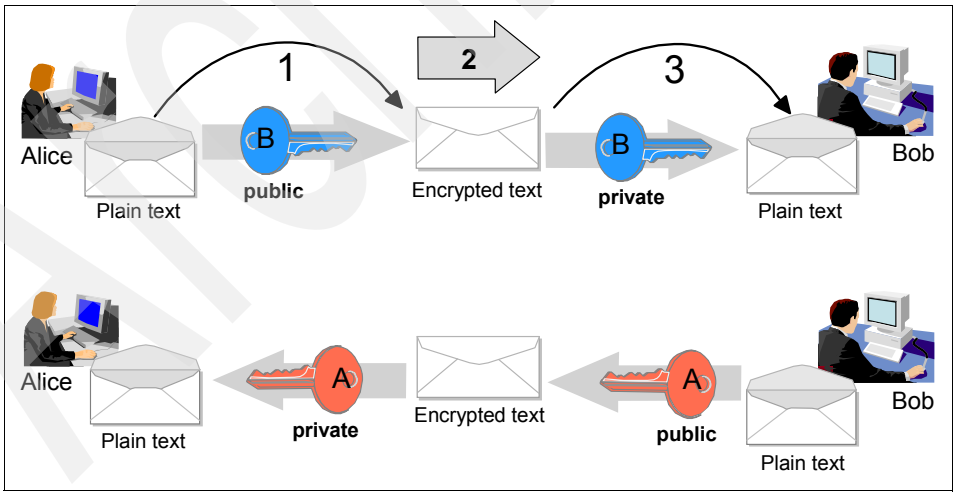


Figure 2-3 Using private key cryptography

Figure 2-3 on page 14 shows a sample communication between two persons: Alice and Bob.

1. Alice wants to communicate with Bob but she does not want anybody to read the messages. She will use Bob's public key to encrypt the message.
2. Alice sends the message to Bob.
3. Bob uses his private key to decrypt the message.

If Bob wants to answer, he should use Alice's public key for encryption.

This example is not suitable for the encryption of large amounts of data, because public key algorithms are very slow. We use the secure key algorithms to transmit large amounts of data. The session keys must be delivered with the public key algorithm and will be used during the communication.

This is the concept that SSL follows to establish a secure communication.

Certificates

A certificate is a document from a trusted party that proves a person's identity. PKI certificates work in a similar way: If someone has a certificate from a trusted party, we can make sure of his or her identity.

Signatures

Signatures also work as in everyday life. For signatures used in the PKI environment, the information encrypted with a person's (the sender) private key is unique to this person. Anybody can decode the message, and the source will be identified, because only one public key can open the message: the sender's public key. This message is almost good enough to be used for a digital signature; the only problem is that we would like to sign documents, and an encrypted document is too long to be a signature.

Signatures are not enough for identification. For example, when someone wants to travel by air, a passport has to be shown as proof of identification. The certificate, similar to a passport, is issued by a trusted authority. It should contain information about the owner and should be signed by the authority.

The standard that defines the form of a certificate is called X.509. This standard also defines the attributes of a certificate, for example: X.500 name, issuer's name, distinguished name, serial number, and so on.

Elements of a certification authority system

A PKI system completes the tasks related to public key cryptography. These tasks should be separate, meaning that a PKI system should have some well-defined units to execute the different tasks. In some cases, the PKI implementation must separate the different functions physically (for example, in a commercial CA system). In this case, the elements listed next are located on different servers.

The logical elements of a PKI system are:

- Certificate Authority (CA)
- Registration Authority (RA)
- Certificate Repository (CR)

Certificate Authority (CA)

The CA component is the heart of a PKI system; it provides the “stamp” to the certificate. In some implementations, the CA component is issued with the Registration Authority component. It stores its private key and can sign the certificate requests with it. This private key should be kept in a very secure place. If this key is corrupted, the whole certification tree will be unusable. This key can be stored on separate hardware.

Registration Authority (RA)

This component is responsible for the registration process. It is an optional component of a PKI system but, in most cases, it is implemented. The main RA task is the verification of client requests.

Certificate Repository (CR)

This component is often called a *certificate directory*. The users of a PKI system use the issued certificates to authenticate themselves. When someone receives a signed message, the receiver checks the signature. If the signature was issued by a trusted party, the message is considered a trusted message. Otherwise, there is a problem. The certificate could have been revoked (for example, the owner left the company), so it should not be considered trusted. This problem is solved by publishing certificates in the certificate repository. When a user receives a message with a certificate, the validity of the certificate can be verified.

The list of revoked certificates is called Certificate Revocation List (CRL) and is usually stored in the CR. The most common way to implement a CR is to use the Lightweight Directory Access Protocol (LDAP) standard (RFC2587).

Certification process

Usually, there are two methods to issue certificates. The difference between the processes is the location where the client's private key will be generated.

- In the first case, the client key pair is generated on the client side (on the client machine). The client creates a certificate request, which contains some information about the client (public key, name, e-mail address, key usage, some optional extensions, and so on). The request is signed with the private key of the client and sent to the server. The server identifies the client before issuing the certificate. The first step is to verify whether the signature at the end of the request is valid. (The public key in the request can be used for validation.) If no error is encountered, then either the certificate can be issued or another client validation process can be started. The most secure method of client validation is for the client to appear personally and certify themselves at the authority location. If the client certification is successful, the certificate for the public key is created with the desired key usage. The client can download the certificate into his or her browser registry or onto a smart card.
- The other way to issue certificates is to execute the key generation process on the server side. This means that private keys are created on the server side. This solution presents some problems:
  - Key generation requires a lot of computing power. There should be very powerful computers applied as Certificate Authority (CA) machines or key generation will be very slow (in case of multiple requests).
  - The private key must be issued and sent to the client, creating a weak point in the security.

  In some situations, this method is better for issuing certificates. For example, a research institute with a few hundred employees wants to make the entrance of the building more secure and wants the computers to be used by appropriate people. The company decides to use smart cards for solving both problems. A PKI system can be implemented and every employee can get a smart card with a certificate and a private key. Obviously, the company will not establish a Web registration module for the employees (because of the fixed and small number of certificates to issue), but it will create the keys and certificates, install them on the cards, and issue the cards to the employees. This process does not have any weak points, because the cards are given personally to each person. Smart cards usually do not allow the exporting of private keys, so they cannot be corrupted (unless the card is stolen).

Self-signed certificates

Self-signed certificates can be used in a trusted environment in which the two parties do not need a third party to certify them. To ensure the trust between the two parties, the certificates are exchanged between the two in a secure manner, prior to any contact.

Self-signed certificates are convenient in internal applications and intranet environments. They are not sufficient when the parties cannot exchange certificates in a secured, trusted manner, or when the communication happens ad hoc between two parties who do not know about each other.

Infrastructure

A Public Key Infrastructure (PKI) system acts as a trusted third-party authentication system, issuing digital certificates for the communication parties (users and applications). Some of its tasks are:

- Issuing certificates
- Revoking certificates
- Renewal of certificates
- Suspension and resumption of certificates
- Management of issued certificates
- Issuing a list of revoked certificates
- Protection of the private key

Figure 2-4 shows three certification scenarios.

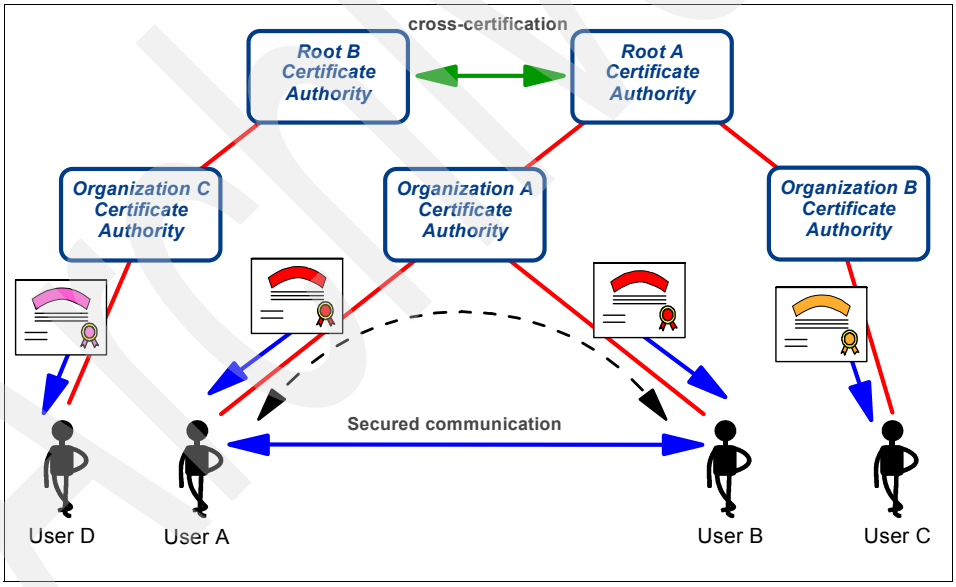


Figure 2-4 Simple certification scenarios

The depicted certification scenarios are:

- When User A wants to talk to User B, both of their certificates are issued and signed by the same Certificate Authority (Organization A); they can trust each other, and the secure communication is built based on the trust.

- When User A or User B wants to talk to User C, their certificates come from the same Root Certificate Authority (Root A); they can trust each other again. This scenario shows the hierarchy of the certificates, where the certificate has been signed by a chain of CAs. As long as the two parties have mutual Certificate Authorities along the line, they can trust each other.
- When User D wants to talk to User A or User B or User C, their certification paths are different. To resolve the problem, the two root Certificate Authorities (Root A and Root B) can create a trust between themselves by setting up a cross-certification. When the two parties have cross-certified CAs along the path, they can trust each other.

2.5 Kerberos

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

> "The Kerberos protocol uses strong cryptography so that a client can prove its identity to a server (and vice versa) across an insecure network connection. After a client and server have used Kerberos to prove their identity, they can also encrypt all of their communications to assure privacy and data integrity as they go about their business." (from the MIT Kerberos Web site)

You can find more information about Kerberos at:

`http://web.mit.edu/kerberos/www/`

2.6 Firewall

Firewalls are key network infrastructure components in security. They have many functions that help to separate network segments and provide services to connect them.

By separating network segments, the communication can be controlled on the protocol level between clients and servers.

A few security functions that firewalls can provide:

- Hide actual server names and addresses from outside connections.
- Filter communication based on originating addresses.
- Filter communication based on protocols.
- Authenticate connecting clients.

You can find numerous documents about firewalls on the Web.

3

Security fundamentals for J2SE, J2EE, and WebSphere

This chapter is about security fundamentals in Java 2 Platform, Standard Edition (J2SE), Java 2 Platform, Enterprise Edition (J2EE), and particularly IBM WebSphere Application Server.

The purpose of this chapter is to give you a high-level overview about almost everything that is related to security in the scope we defined in the title. We do not cover everything in depth, but at least mention everything and give you a reference for further research.

3.1 Introduction

This may have a strong influence on the WebSphere security configuration (that is stored in the file system) and the overall application runtime security environment.

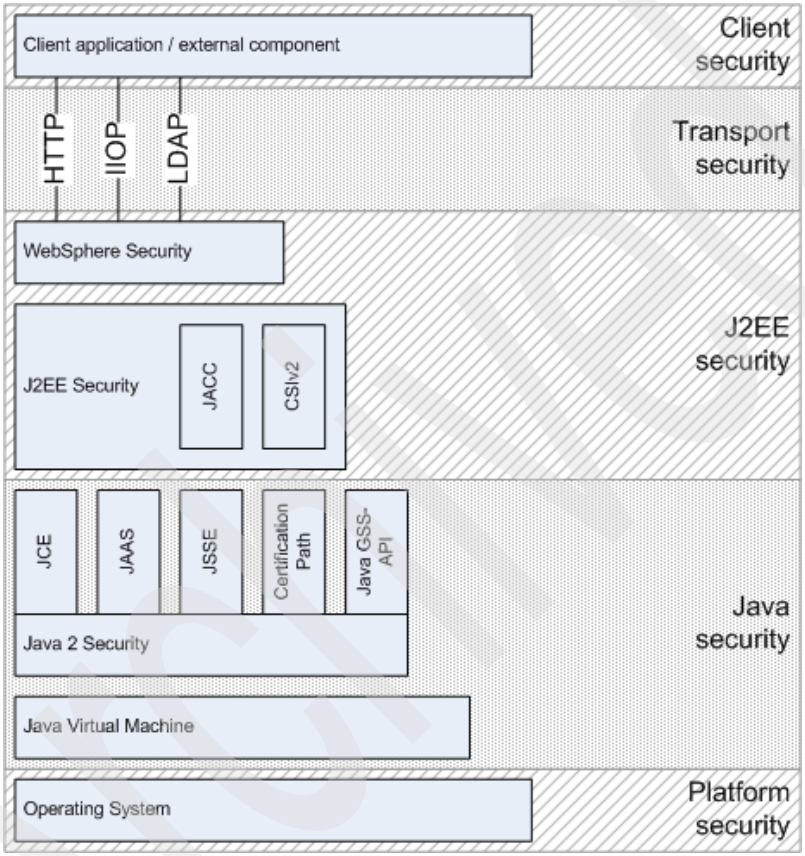


Figure 3-1 WebSphere environment security layers

WebSphere Application Server security sits on top of operating system security and the security features provided by other components, including the Java language, as shown in Figure 3-1.

- Operating system security should be considered in order to protect sensitive WebSphere configuration files and to authenticate users when the operating system user registry is used for authentication. This is extremely important in a distributed WebSphere environment when potentially different operating systems and different user registries might be involved. Keeping users (and

their passwords) and groups in sync across many different machines might be a problematic administration task.

- Standard Java security is provided through the Java Virtual Machine (JVM) used by WebSphere and the Java security classes.
- Java 2 security enhances standard JVM security by introducing fine-grained access, easily configurable security policy, extensible access control structure, and security checks for all Java programs (including applets).
- Common Secure Interoperability (CSIv2) protocol adds additional security features that enable interoperable authentication, delegation and privileges in CORBA environment. It supports interoperability with EJB 2.0 specification and can be used with SSL.
- J2EE security uses the security collaborator to enforce J2EE-based security policies and support J2EE security APIs. APIs are accessed from WebSphere applications to access security mechanisms and implement security policies.

WebSphere Application Server V5 security relies on and enhances all of these layers. It implements security policy in a unified manner for both Web and EJB resources.

You can find more information about Java-related security at:

http://www.ibm.com/developerworks/java/jdk/security/142/

3.2 Java 2 security

Earlier Java implementations, prior to Java V1.2, only had the sandbox model, which provided a very restricted environment. With Java V1.2, a new security model has been introduced. Figure 3-2 on page 24 shows the new security model for Java V1.2.

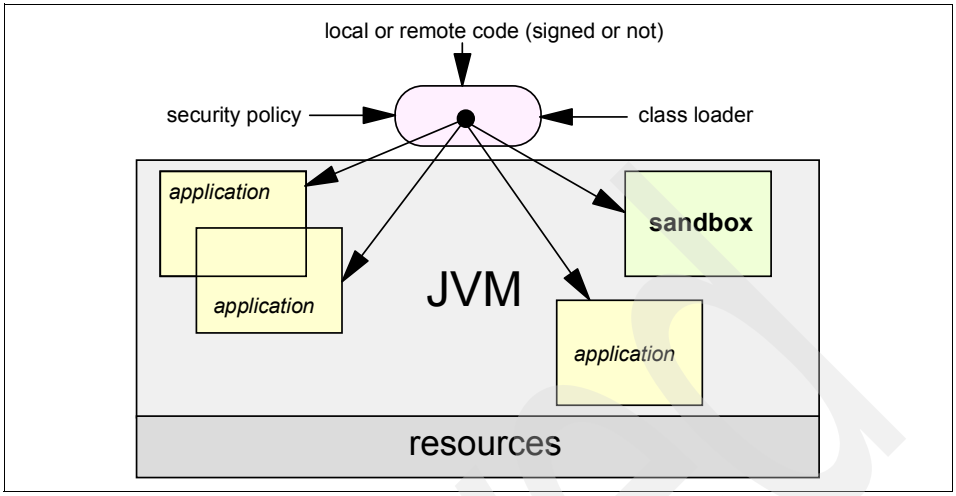


Figure 3-2 Java 2 platform security model

The new model is meant to provide the following security features for the JVM:

- Fine-grained access control: This was available in the earlier version using programmatic access control security.
- Easy configuration of security policy: It also was available in previous versions, and also used programmatic security.
- Easy extension for the access control structure: The new architecture allows typed security permissions and provides automatic handling for them.
- Extension of security checks to all Java programs (both applications and applets): Every Java code is under security control, which means that local code is no longer trusted by default.

The fundamental concept and an important building block in system security is the *protection domain*.

Definition: A *domain* can be scoped by the set of objects that are directly accessible by a principal (an entity in the computer system to which permissions are granted). Classes that have the same permissions but are from different code sources belong to different domains.

A *principal* is an entity in the computer system to which permissions (and as a result, accountability) are granted.

(From the *Java 2 Platform Security Architecture V1.0* paper by Sun Microsystems)

There are two distinct categories of protection domains:

- System domain
- Application domain

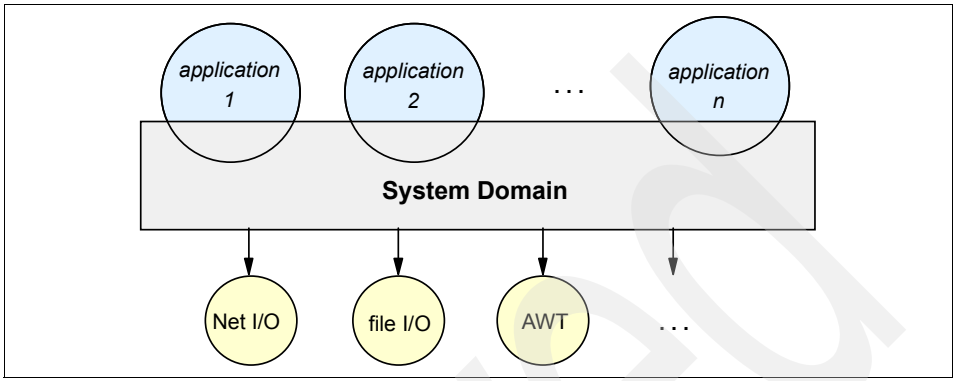


Figure 3-3 Protection domains

Protection domains are determined by the policy currently in effect. The Java application environment maintains the mapping between code, their protection domains, and their permissions.

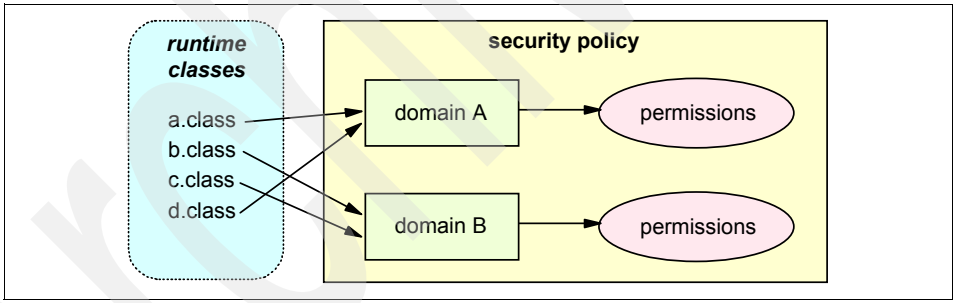


Figure 3-4 Class-domain-permission mapping

Generally, a less-"powerful" domain cannot gain additional permissions as a result of calling or being called by a more-powerful domain.

The *doPrivileged* method can be used to call a piece of trusted code to temporarily enable access to more resources than are available directly to the application. This method is capable of making a call to trusted code in a programmatic way.

Security management

The security manager defines the outer boundaries of the Java sandbox. The customizable security manager establishes custom security policies for an application. The concrete *SecurityManager* provided with Java V1.2 enables you to define your custom policy, not in Java code but in an ASCII file called the *policy file*.

The security manager is not loaded automatically when an application runs. To activate the manager, the user must specify this command-line argument for the Java runtime:

```
-Djava.security.manager
```

A custom security manager class can be also specified in the command line:

```
-Djava.security.manager=com.mycompany.MySecurityManager
```

If nothing is specified, then the default security manager will be initialized for the application.

Access control

The *java.security.ProtectionDomain* class represents a unit of protection within a Java application environment and is typically associated with a concept of *principal*.

The java.security.AccessController class is used for the following purposes:

- To decide whether access to a critical resource is allowed or denied, based on the security policy currently in effect
- To mark code as being *privileged*
- To obtain a *snapshot* of the current calling context to support access-control decisions from a different context

Any code that controls access to system resources should invoke AccessControler methods if it wishes to use the specific security model and access control algorithm utilized by these methods.

Security permissions

The permission classes represent access to system resources. The java.security.Permission class is an abstract class and is subclassed to represent specific accesses.

Permissions in Java V1.2 are:

- java.security.Permission

  This abstract class is the ancestor of all permissions.

- java.security.PermissionCollection

  This holds a collection of the same type of permissions (homogeneous).

- java.security.Permissions

  This holds a collection of any type of permissions (heterogeneous).

- java.security.UnresolvedPermission

  When the policy is initialized and the code that implements a particular permission has not been loaded or defined in the Java application environment, in this case the UnresolvedPermission holds the unresolved permissions.

- java.security.UnresolvedPermissionCollection

  This holds a collection of UnresolvedPermissions.

- java.io.FilePermission

  This holds permission definitions for file resources. Actions on a file can be read, write, delete, execute.

- java.security.SocketPermission

  This permission represents access to network sockets; actions on a socket can be: accept, connect, listen, resolve.

- java.security.BasicPermission

  This extends the Permission class and can be used as the base class for other permissions.

- java.util.PropertyPermission

  This class targets the Java properties as set in various property files; actions can be read and write.

- java.lang.RuntimePermission

  The target for this permission can be represented by any string and there is no action associated with the targets.

- java.awt.AWTPermission

  Similar to the previous permission, but it is related to targets in the Abstract Window Toolkit (AWT).

- java.net.NetPermission

  This controls the Net-related targets; no actions associated.

- java.lang.reflect.ReflectPermission

  This is a Permission class for reflective operations. It has no actions; it works like the RuntimePermission.

- java.io.SerlializablePermission

  This controls the serialization related targets; no actions associated.
- java.security.SecurityPermission

  This controls access to security-related objects; no actions associated.
- java.security.AllPermission

  This permission implies all permissions.

Policy files

The policy can be specified within one or more policy configuration files that indicate what permissions are allowed for codes from specified code sources.

Definition: A policy configuration file essentially contains a list of entries. It may contain a *key store* entry, and contains zero or more *grant* entries.

The key store can be defined according to the following grammar:

```
keystore "keystore_URL", "keystore_type";
```

A grant entry can be defined according to the following grammar:

```
grant [SignedBy "signer_names"] [, CodeBase "URL"] {
   permission permission_class_name [ "target_name" ] [, "action"]
[,SignedBy "signer names"];
   ...
};
```

Each grant entry consists of a CodeSource and its permissions, where a CodeSource consists of a URL and a set of certificates and the grant entry includes a URL and a list of signer names.

Property expansion is possible in the policy files and in the security properties file.

Example 3-1 Sample policy file

```
keystore "c:\keystores\mykey.jks", "jks"

grant codeBase "http://java.sun.com/*", signedBy "WebDeveloper" {
   permission java.io.FilePermission "/files/*", "read";
   permission java.io.FilePermission "${user.home}", "read,write";
}
```

When the JVM loads a new class, the following algorithm is used to check the policy settings for that particular class:

1. Match the public keys, if code is signed.
2. If a key is not recognized in the policy, ignore the key. If every key is ignored, treat the code as unsigned.
3. If the keys are matched or no signer was specified, try to match all URLs in the policy for the keys.
4. If either key or URL is not matched, use the built-in default permission, which is the original sandbox permission.

Policy files in runtime

The following list shows how the policy files can be specified for a Java runtime and where those policy files are located:

- System policy file:

  {java.home}/lib/security/java.policy

- User policy file:

  {user.home}/.java.policy

- Policy file locations are also specified in the security properties file:

  {java.home}/lib/security/java.security

- You can specify an additional or different policy file when invoking execution of an application using the appropriate command line arguments; for example:

  ```
  java -Djava.security.manager -Djava.security.policy=MyPolicyURL
  MyApplication
  ```

  When the policy file is specified using double equals, the specified policy file will be used exclusively; for example:

  ```
  -Djava.security.policy==MyOnlyPolicyURL
  ```

Security exceptions

The following exceptions ship with the Java V1.2 SDK:

- java.security.SecurityException

  This exception and its subclasses should be runtime exceptions (unchecked, not declared) that are likely to cause the execution of a program to stop. Such an exception is thrown when a security violation is detected (for example, when trying to access an unauthorized resource).

- java.security.GeneralSecurityException

  This is a subclass of java.lang.Exception (must be declared or caught) that is thrown in other cases. Such an exception is thrown when a security-related (but not vital) problem is detected, such as passing an invalid key.

Secure class loading

Dynamic class loading is one of the strengths of the Java platform because it provides the ability to install components at runtime. It is also critical in providing security because the class loader is responsible for locating and fetching the class file, consulting the security policy, and defining the class object with the appropriate permissions.

The *java.security.SecureClassLoader* is a subclass and an implementation of the abstract *java.lang.ClassLoader* class. Other classloaders subclass the SecureClassLoader to provide different class-loading facilities for various applications.

Debugging security

Use the *-Djava.security.debug=access,failure* argument in the virtual machine. This flag dumps the names of failing permission checks.

For example: Start with minimal security permissions, then run a test and check which permissions are failing. Add the necessary permissions to the policy file, then run your test again for re-checking. Repeat these steps until you have set all of the necessary permissions. This only helps you to identify the permissions you have to set; it does not help to find the right settings for the permissions.

For more about Java 2 security, refer to the official Java Sun Web site at:

`http://java.sun.com/security/index.jsp`

3.3 Basic cryptography

J2SE V1.4 provides basic cryptography functions and API for developers. With the cryptography API, developers can encrypt and decrypt information programmatically in their applications.

Java Cryptography Extension (JCE)

The JCE offers a framework and implementations for encryption, key generation and key agreement, and algorithms for Message Authentication Code (MAC). It supports encryption through symmetric, asymmetric, block, and stream ciphers, and supports secure streams and sealed objects. JCE is a supplement to the

Java 2 platform, which already has interfaces and implementations of message digests and digital signatures.

You can find more information about JCE at:

http://java.sun.com/products/jce/

3.4 Authentication

J2SE V1.4 provides authentication services. Programmers can use the service to implement authentication for their applications, or use the SPI to extend the existing service with new modules and functions.

Java Authentication and Authorization Service (JAAS)

JAAS provides the ability to enforce access controls based on who runs an application. Traditionally, Java 2 provided codesource-based access controls (access controls based on where the code originated and who signed it), but lacked the ability to enforce access controls based on who ran the code.

You can find more information about JAAS at:

http://java.sun.com/products/jaas/

3.5 Authorization

JAAS also provides authorization services on the J2SE level for programmers and for applications. The Java Authorization Contract for Containers (Java ACC) is a specification for externalizing authorization decisions and delegating them from the container to an external application.

Java ACC

The Java ACC specification defines a contract between J2EE containers and authorization policy modules such that container authorization functionality can be provided as appropriate to suit the operational environment. It defines how external authorization providers are to be interfaced with J2EE containers.

You can find more information about JACC at the following URL:

http://java.sun.com/j2ee/javaacc/index.html

3.6 Secure connection

JSSE (Java Secure Socket Extension) is part of J2SE. It provides an API that developers can use to set up secure connection for the network transport to secure communication protocols.

JSSE

The JSSE, a Java package, enables secure Internet communications by implementing a Java version of SSL (Secure Sockets Layer) and TLS (Transport Layer Security) protocols. It also is involved with data encryption, server authentication, message integrity, and optional client authentication.

You can find information about JSSE at:

http://java.sun.com/products/jsse/

3.7 Security context

The APIs supporting authentication, authorization, and secure communication require additional APIs to handle and maintain security-related objects and contexts.

Certification Path API

The Java Certification Path defines a set of classes and interfaces for creating, building, and validating digital certification paths. A digital certificate is a data structure of the binding between a subject and a public key signed by a Certification Authority (CA).

You can find more information about Certificate Path API at:

http://java.sun.com/j2se/1.4.2/docs/guide/security/certpath/CertPathProgGuide.html

Java GSS-API

With JGSS (Generic Security Services) messages can be exchanged securely between applications. The Java GSS-API holds Java bindings for the Generic Security Services Application Program Interface (GSS-API), which is defined in RFC 2853. Application programmers use GSS-API for uniform access to security services on a variety of underlying security mechanisms, including Kerberos.

You can find more about Java GSS-API at:

http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/index.html

You can find information about the Generic Security Services Application Program Interface (GSS-API) in the RFC 2853 at:

http://www.ietf.org/rfc/rfc2853.txt

3.8 Web security

We discuss a few aspects of Web security in this section. Some Web-related security functions and mechanisms are so common that they deserve to be discussed here.

3.8.1 Web authentication mechanisms

Web servers and application servers support numerous different authentication methods of retrieving the user's identity. Web clients, including numerous browser and Web service clients, also support different authentication mechanisms to provide the user's identity to the server. This section discusses the most common authentication mechanisms available today in Web servers and clients.

Basic authentication

Basic authentication is the fundamental authentication method. When the server requests user authentication, the client sends a user name and password encoded in the HTTP header (no encryption). After initial authentication, the Web client keeps sending the user name and password until the end of the session.

Digest

Basic authentication is a vulnerable mechanism and is not safe. With the digest mechanism, the client only provides a digest value (an MD5 checksum by default) of the user name, password, requested URI, and HTTP method to the server. For more information, read RFC 2069 at:

http://www.ietf.org/rfc/rfc2069.txt

Form

Form-based logon is a popular method for the logon process. This mechanism uses the standard HTTP POST method to send logon information to the application server. Every application server has its own implementation and naming constraints for the fields and for the URI to submit to. It is a fairly secure method if the logon information is sent over a secured HTTPS connection.

Certificate-based authentication

User certificates can be stored on the client side. Most browsers have the capability to store and present certificates when required. Besides the encryption keys, certificates also store user information that can uniquely identify a subject. Application servers can extract the user information from certificates and use it to authenticate the user.

NTLM

NTLM is an authentication protocol used in various Microsoft® network protocols. It follows a challenge-response mechanism for authentication. A few Web browsers support this mechanism. Explaining the protocol and how it is used for authentication is beyond the scope of this paper, but you can find numerous documents on the Web to find out the details.

SPNEGO

Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) is a negotiation protocol used for authentication, defined in RFC 2478. SPNEGO enables Web clients to use the Kerberos authentication mechanism over HTTP by wrapping the Kerberos token inside an SPNEGO token. Because the Kerberos authentication mechanism is a multi-phase interaction, SPNEGO also defines a negotiation protocol for authentication. Only a few Web browsers and Web servers support SPNEGO.

You can find the first of a series of three articles about SPNEGO at:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecure/html/http-sso-1.asp

Single Sign-On

Although Single Sign-On is not an authentication mechanism, it is an integral part of the logon process on the Web. Web clients might access many different Web applications during a session in a single security domain (such as one company or one business). Every time the user accesses a new application, the system must identify the user. If there is no mechanism in place to address this process, the user has to authenticate every single time. Single Sign-On is a mechanism that makes logon convenient and prevents users from having to authenticate more than once in the same security domain.

Some of the many Single Sign-On mechanisms for Web applications include:

- LTPA token (IBM proprietary security token)
- SPNEGO, which uses Kerberos tokens in HTTP
- Security reverse proxy server, such as WebSEAL

3.9 Security tools

SSL relies on the existence of digital certificates. A digital certificate reveals information about its owner, such as identity. During the initialization of an SSL connection, the server must present its certificate to the client in order for the client to determine the server's identity. The client may also present the server with its own certificate for the server to determine the client's identity. Therefore, SSL is a means for propagating identity between components.

3.9.1 ikeyman

ikeyman is the primary tool from IBM for managing key stores.

Note: The ikeyman used in this section, ikeyman version 7.0.3.7, is shipped with WebSphere Application Server V6.

Changing the supported key store formats

The ikeyman that is installed on your system does not have all of the necessary key store formats enabled by default. The reason is that the supported key store types are defined in the Java runtime. By default these key stores are available:

- JKS
- JCEKS
- PKCS12

Use the KDB (CMS) type with IBM HTTP Server. You can easily enable this type of key store by simply editing the java.security file under the <IHS_home>/_jvm/jre/lib/security directory. Find the part of the file where the security.provider items are listed, and add this line at the end of the list:

```
security.provider.6=com.ibm.spi.IBMCMSProvider
```

Note that the number has to be the next available number in the list.

Start ikeyman

Use a command line to start ikeyman from either WebSphere's bin directory or IBM HTTP Server's bin directory. Look for the **`ikeyman`** executable file. After starting the utility, you should find a GUI application on your desktop.

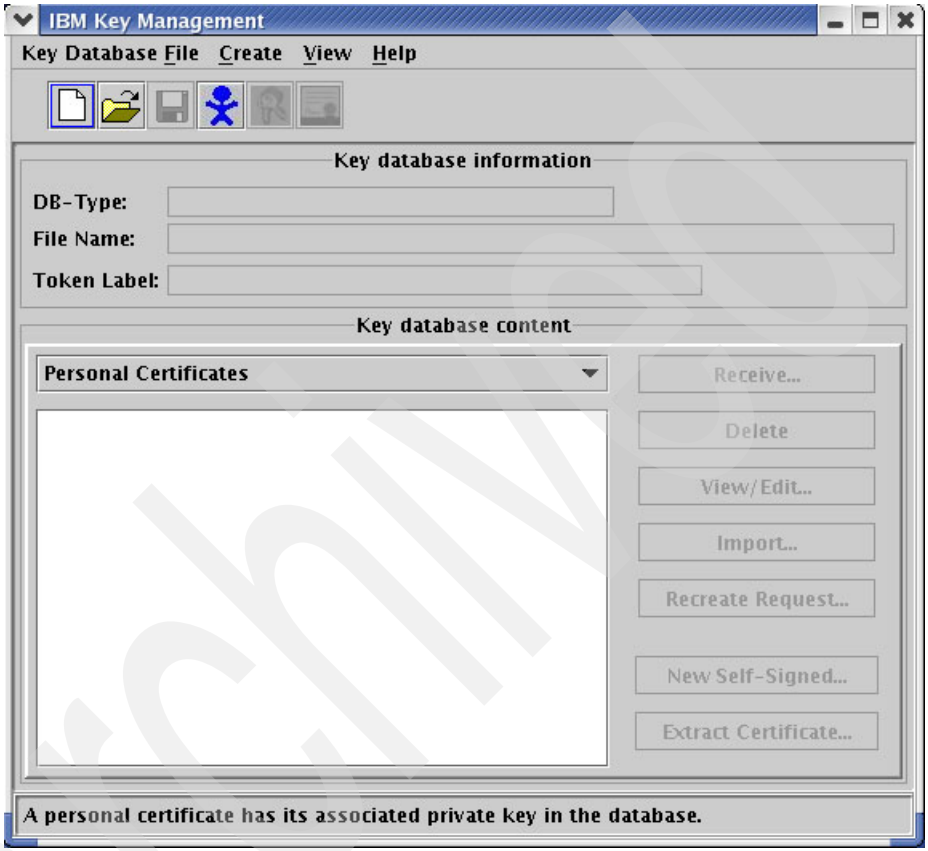


Figure 3-5 ikeyman started on Red Hat Linux®

Create a new key store

The first step is to create a new key store:

1. Select **Key Database File** → **New** from the menu to open a new dialog.
2. Define the key database type and the file name.

   You should have a directory where you store the key store files for your applications. WebSphere Application Server stores the keys under the <WebSphere_root>/etc directory. IBM HTTP Server does not have a specific directory.

3. Click **OK**.

The key store has a set of public certificates already inserted for your convenience. You can find them under the Signer Certificates section. The certificates are from a few major commercial Certificate Authorities, including VeriSign, Entrust, and Thawte.

Managing a self-signed certificate

ikeyman is capable of managing self-signed certificates. You can easily generate the private and public key pairs yourself and export the public certificate so you can use it with other peers:

1. Make sure you have followed the steps from the previous section, “Create a new key store” on page 36, to create a new key store.
2. Select **Create** → **New Self-Signed Certificate** to open a new dialog.
3. Fill out the information to create your certificate.

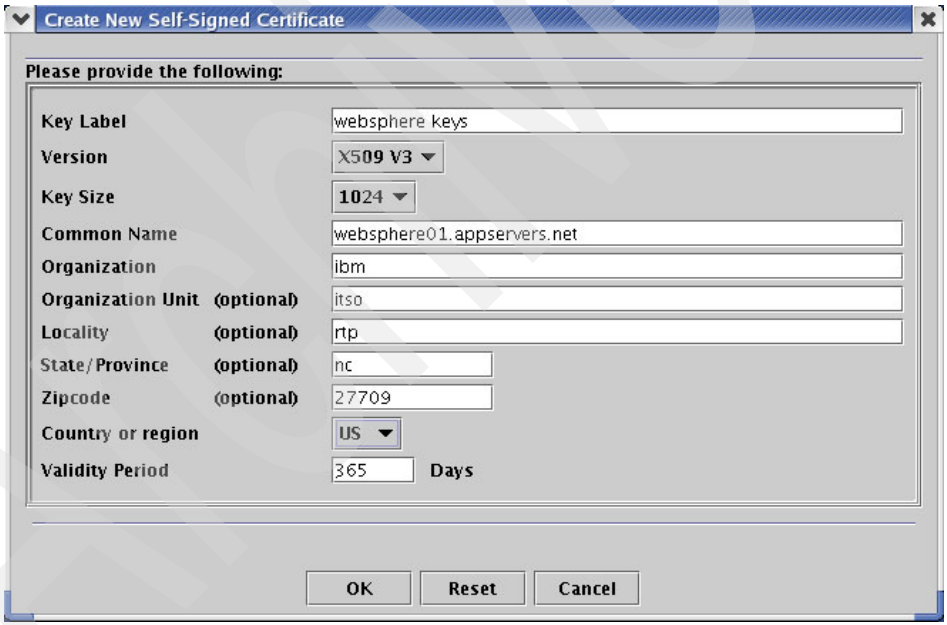


Figure 3-6 Creating a new Self-signed certificate

4. Click **OK**.

You will see the new item in ikeyman under the Personal Certificates section.

To use self-signed certificates between parties, you have to exchange the public certificates. (See “Exchanging public certificates” on page 39.)

Managing a real (non-Self-Signed) certificate

ikeyman can also manage real (non-self-signed) certificates. You can import and export real certificates, and use the tool to generate a certificate request to send to a Certificate Authority.

1. Make sure you have followed the steps in “Create a new key store” on page 36, to create a new key store.
2. Select **Create** → **New Certificate Request** to open a new dialog.
3. Enter the appropriate information. Note that you have to provide a file location where the certification request will be stored. Later you will have to send this file or the content to the Certificate Authority.

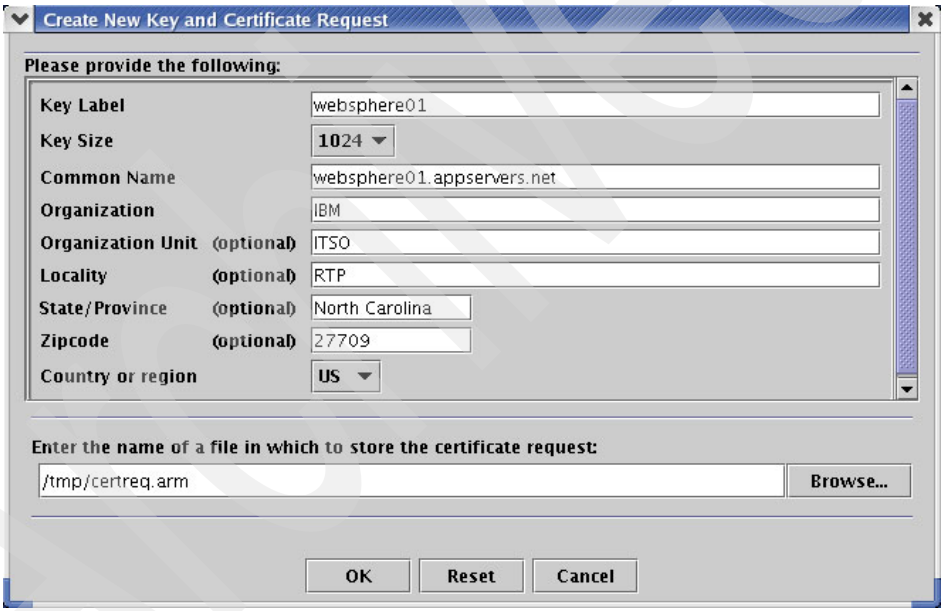


Figure 3-7 Creating a new certificate request

4. Click **OK**.

   The new item is shown in ikeyman under the Personal Certificate Requests section. Meanwhile, a new certificate request file has been generated at the specified location.

5. Use the certificate request file to apply for a certificate at a Certificate Authority. This is a short and easy process that you can do it over the Internet.

The result is either a file or a piece of encoded text that you copy into a file (for example: `response.arm`).

6. When the file is available, switch to the **Personal Certificates** section in ikeyman, then click **Receive**, and a new dialog opens.
7. Enter the location of the file that holds the response from the Certificate Authority and click **OK**.
8. ikeyman matches up the request entry and the response, removing the request and inserting a new personal certificate. Select the item and click **View/Edit** to see the details of the signer Certificate Authority under the Issued by section.

Close the key store file

To start using the key, you have to close the key store or close ikeyman.

Best practice for key stores

Key stores can hold multiple key and certificate entries in one file. WebSphere distinguishes two different key stores when configuring security:

- Key store

  You keep your private keys in the key store. These keys are either self-signed or issued by a Certificate Authority.

- Trust store

  The trust store keeps the public key of your own key pair and the public certificates of the parties you trust.

It is wise to keep the two different key stores and keys and certificates separate. After you have issued or received your own key pair, store the private key locked with a password and never open it again. The trust store might be changed occasionally to add a new certificate you trust.

Exchanging public certificates

This section provides details and step-by-step instructions for exchanging public certificates between two key stores or trust (certificate) stores. You must perform the certificate exchange when you want to set up trust between two parties based on certificates. Usually you use this process with self-signed certificates because real certificates issued by well-known Certificate Authorities are already included in the key and trust stores.

WebSphere Application Server demo key stores

WebSphere Application Server provides a set of certificates that may be used for testing purposes. The identities in the certificates are generic and the expiration dates are set artificially low. This section describes the process for creating digital certificates tailored for use in a production system.

WebSphere supports the concept of two types of key store: a *key file* and a *trust file*. A key file contains a collection of certificates and the associated private key for each certificate. A server manages at least one key file, although a client may also manage one. A trust file contains a collection of certificates that are considered trustworthy and against which the presented certificate will be matched during an SSL connection initiation in order to assure identity. A client typically manages at least one trust file, although a server may also manage one (see Figure 3-8).

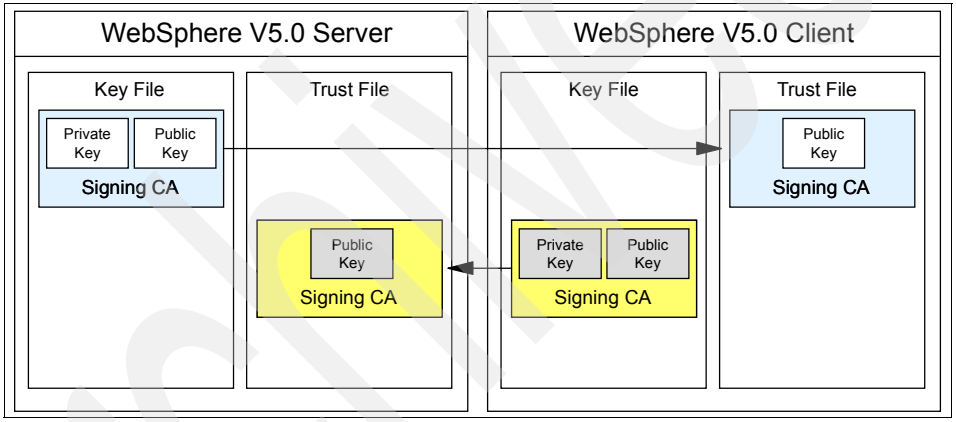


Figure 3-8 Correlation between server and client key stores

This demonstrates how the two types of key store may be used, but remember that it is also possible to combine the key and trust files. WebSphere provides the following key stores in the <WebSphere_root>/profiles/<server_profile>/etc directory.

You can see that the public certificates are shared between the two parties' key stores. The server's public certificate from the key store is imported into the client's trust store. The client's public certificate from the key store is imported into the server's trust store. This last one is required for scenarios where mutual authentication is required.

Table 3-1 WebSphere default key stores

File	Description
DummyServerKeyFile.jks	Server-based key file
DummyServerTrustFile.jks	Server-based trust file
DummyClientKeyFile.jks	Client-based key file
DummyServerTrustFile.jks	Client-based trust file

The key store type in this case is Java Key Store (JKS), a format that is supported by both WebSphere and the supplied key generation utility, ikeyman. This utility is used in the next section to generate a new certificate.

You can open the demo key store files and investigate the contents or export the certificates yourself. The password for the demo key stores is `WebAS` (case sensitive).

Exchanging certificates in ikeyman

This section provides details about exchanging public certificates between two parties, for example, between EJB client and EJB container, Web server plug-in and WebSphere Web container, or Web browser client and Web server.

We assume that there are four key store files, similar to the demo key files presented in the previous section. We follow the practice of using two key stores.

- ServerKey.jsk for storing the private keys for the server.
- ServerTrust.jsk for storing the public certificate for the server keys.
- ClientKey.jks for storing the private keys for the client.
- ClientTrust.jsk for storing the public certificate for the client keys.

Whether the certificates are real or self-signed does not matter in this case. We also use .jks key stores here, but the same method can be applied to other type of key stores too, such as CMS (.kdb) key database.

1. Export the public certificate from the server trust store:
   a. Start ikeyman, and open the **ServerTrust.jks** file.
   b. Switch to the Signer Certificates section, then select the **XYZ server public certificate** alias.
   c. Click **Extract** to export the certificate. Provide the details for saving the certificate, for example: `/tmp/exchange/xyzserverpublic.arm` and use the data type Base64 encoded ASCII data.

2. Import the server's public certificate into the client's trust store:
   a. Open the **ClientTrust.jks** file in ikeyman. You may have to start ikeyman on the client machine if the trust store is not distributed from the server.
   b. Switch to the Signer Certificates section, then click **Add** to import the certificate.
   c. Browse for the **xyzserverpublic.arm** file that you exported previously. After clicking **OK**, provide a label for the new entry (alias), for example: `xyz server public`. The new certificate should appear in ikeyman.

      You may have to move the exported certificate file if the key and trust stores are not on the same machines.
   d. Close ikeyman.
3. At this point, we have set up the certificates for one-way authentication (server authentication). We need to import the client's public certificate to the server's trust store for mutual authentication scenarios, where the client also has to authenticate itself. First, export the client's public certificate:
   a. Start ikeyman, and open the **ClientTrust.jks** file.
   b. Switch to the Signer Certificates section, then select the **ABC client public certificate** alias.
   c. Click **Extract** to export the certificate. Provide the details for saving the certificate, for example: `/tmp/exchange/abcclientpublic.arm` and use the data type Base64 encoded ASCII data.
4. After exporting the client's certificate, we can import it to the server's trust store:
   a. Open the **ServerTrust.jks** file in ikeyman. You may have to start ikeyman on the server machine if the trust store is not distributed from the server.
   b. Switch to the Signer Certificates section, then click **Add** to import the certificate.
   c. Browse for the **abcclientpublic.arm** file that you exported previously. After clicking **OK**, provide a label for the new entry (alias), for example: `abc client public`. The new certificate should appear in ikeyman.

      You may have to move the exported certificate file if the key and trust stores are not on the same machines.
   d. Close ikeyman.
5. Make sure you delete the files you used during the exchange.

The certificates in the key and trust store should look similar to Figure 3-8 on page 40 with different names.

At this point, both server and client know about each other's public certificates, they can authenticate each other, and they can start exchanging information securely.

Exchanging certificates between non-identical key stores

The previous section described how to exchange public certificates between two identical JKS key stores. You can also exchange certificates between two non-identical key stores, such as JKS and KDB. The steps have minor differences from those we just described, but the concept is exactly the same.

3.9.2 J2SE 1.4 security tools

The Java 2 Standard Edition package also provides basic security tools. These tools are explained in this section.

keytool

keytool is a command line tool to manage key stores and associated certificate chains. The main user of the tool is the end user. keytool is the J2SE equivalent of the ikeyman IBM utility, but keytool is a command line utility and has no GUI.

Figure 3-9 is a representation of a key store file with all of the components.

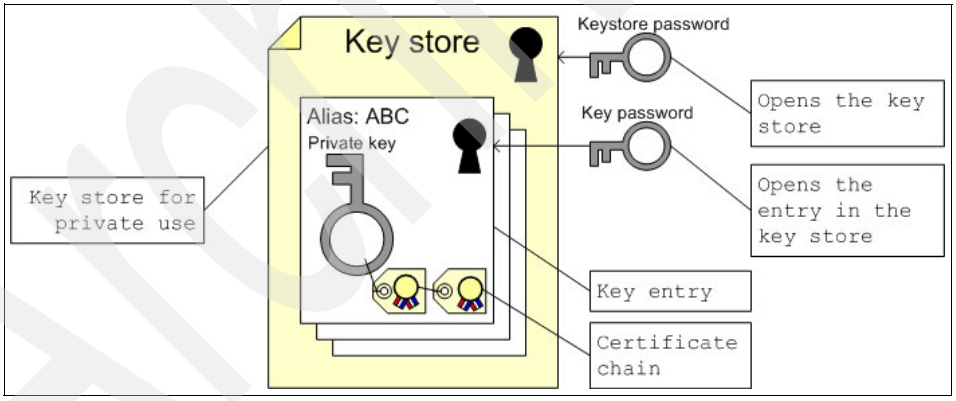


Figure 3-9 Key store for use with private entries

Figure 3-10 shows a key store that holds public certificates (a trust store).

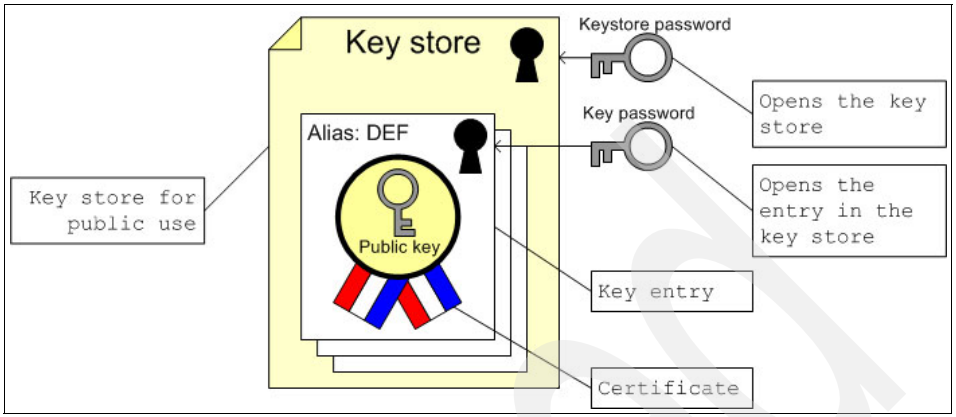


Figure 3-10 Key store for use with public entries

For more information about the keytool application for UNIX® systems, refer to:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html

For information about the keytool application for Windows® systems, visit:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html

jarsigner

The jarsigner tool can generate and verify signatures for Java archives (JAR). Developers can sign their packages and distribute them in a trusted environment. Figure 3-10 on page 44 shows the equation of creating a signed .jar.

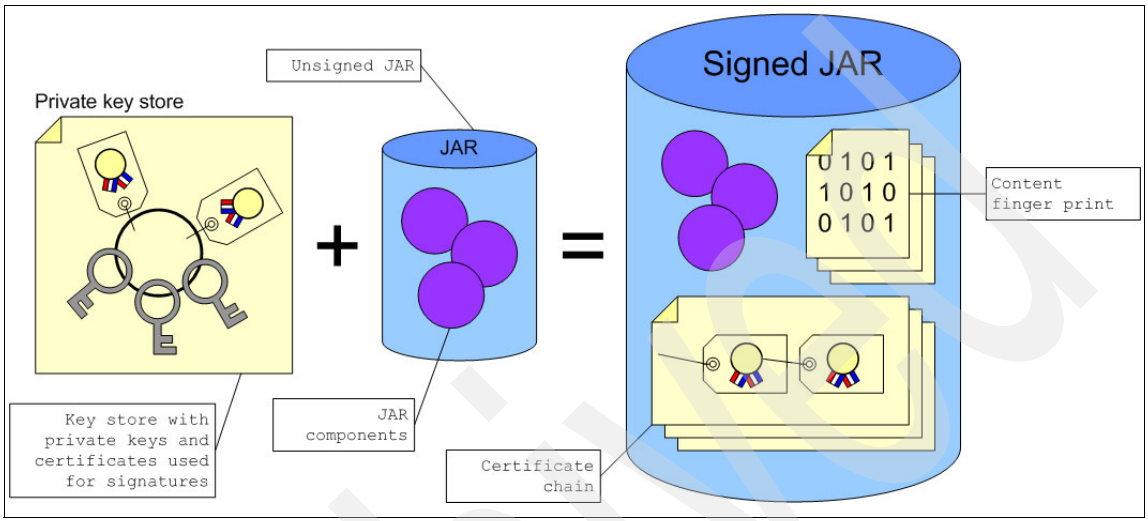


Figure 3-11 The components you will need to sign a .jar

You must have a key store with private keys and certificates to sign a .jar file. The resulting .jar holds the original content and a few extra elements, including:

- A fingerprint of the content to avoid changing it
- A certificate chain that identifies the original signer

For more information about the jarsigner application for UNIX systems, refer to:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/jarsigner.html

For more information about the jarsigner application for Windows systems, see:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/jarsigner.html

policytool

The policytool application is a GUI for editing policy files for the Java runtime.

For more information about the policytool application for UNIX systems, refer to:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/policytool.html

For more information about the policytool application for Windows systems, visit:

http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/policytool.html

Kerberos tools

Kerberos requires a few tools to manage the infrastructure on end-user systems. Three utility tools are provided:

- **`kinit`** helps to obtain Kerberos ticket-granting tickets (TGT) and cache them on the system.

  For more information for UNIX systems, refer to:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/kinit.html`

  For more information for Windows systems, see:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/kinit.html`

- **`klist`** is a utility that can list the Kerberos entries in the cache or in the key table on the local system.

  For more information for UNIX systems, refer to:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/klist.html`

  For more information for Windows systems, see:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/klist.html`

- **`ktab`** manages the Kerberos entries in the key table.

  For more information for UNIX systems, refer to:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/ktab.html`

  For more information for Windows systems, see:

  `http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/ktab.html`

Abbreviations and acronyms

ACL	Access Control List
CA	Certificate Authority
CR	Certificate Repository
CRL	Certificate Revocation List
CSIv2	Common Secure Interoperability, Version 2
GSS-API	Generic Security Services Application Program Interface
GUI	Graphical User Interface
HTTP	Hypertext Transport Protocol
IBM	International Business Machines Corporation
IHS	IBM HTTP Server
IIOP	Internet Inter-ORB Protocol
ITSO	International Technical Support Organization
J2EE	Java 2 Platform, Enterprise Edition
J2SE	Java 2 Platform, Standard Edition
JAAS	Java Authentication and Authorization Service
JACC	Java Authorization Contract for Containers
JAR	Java archives
JCE	Java Cryptography Extension
JKS	Java Key Store
JSSE	Java Secure Socket Extension
LDAP	Lightweight Directory Access Protocol
LTPA	Lightweight Third-Party Authentication
ORB	Object Request Broker
PKI	Public Key Infrastructure
RA	Registration Authority
RFC	Request For Comments
SDK	Software Development Kit
SPNEGO	Simple and Protected GSS-API Negotiation Mechanism
SRPS	Security Reverse Proxy Server
URL	Unified Resource Locator
VM	Virtual Machine

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redpaper.

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 53. Note that some of the documents referenced here may be available in softcopy only.

- *A Secure Portal Using WebSphere Portal V5 and Tivoli Access Manager V4.1*, SG24-6077

  http://www.redbooks.ibm.com/abstracts/sg246077.html

- *AIX 5L Version 5.2 Security Supplement*, SG24-6066

  http://www.redbooks.ibm.com/abstracts/sg246066.html

- *Deploying a Secure Portal Solution on Linux Using WebSphere Portal V5.0.2 and Tivoli Access Manager V5.1*, REDP-9121

  http://www.redbooks.ibm.com/abstracts/redp9121.html

- *Develop and Deploy a Secure Portal Solution Using WebSphere Portal V5 and Tivoli Access Manager V5.1*, SG24-6325

  http://www.redbooks.ibm.com/abstracts/sg246325.html

- *Enterprise Security Architecture Using IBM Tivoli Security Solutions*, SG24-6014

  http://www.redbooks.ibm.com/abstracts/sg246014.html

- *Federated Identity Management with IBM Tivoli Security Solutions*, SG24-6394

  http://www.redbooks.ibm.com/abstracts/sg246394.html

- *IBM WebSphere Everyplace Connection Manager Version 5 Handbook*, SG24-7049

  http://www.redbooks.ibm.com/abstracts/sg247049.html

- *IBM WebSphere V5.0 Security WebSphere Handbook Series*, SG24-6573

  http://www.redbooks.ibm.com/abstracts/sg246573.html

- *Integrated Identity Management using IBM Tivoli Security Solutions*, SG24-6054

  http://www.redbooks.ibm.com/abstracts/sg246054.html

- *Lotus Security Handbook*, SG24-7017

  http://www.redbooks.ibm.com/abstracts/sg247017.html

- *On Demand Operating Environment: Security Considerations in an Extended Enterprise*, REDP-3928

  http://www.redbooks.ibm.com/abstracts/redp3928.html

- *Using LDAP for Directory Integration*, SG24-6163

  http://www.redbooks.ibm.com/abstracts/sg246163.html

- *WebSphere MQ Security in an Enterprise Environment*, SG24-6814

  http://www.redbooks.ibm.com/abstracts/sg246814.html

- *WebSphere Portal Collaboration Security Handbook*, SG24-6438

  http://www.redbooks.ibm.com/abstracts/sg246438.html

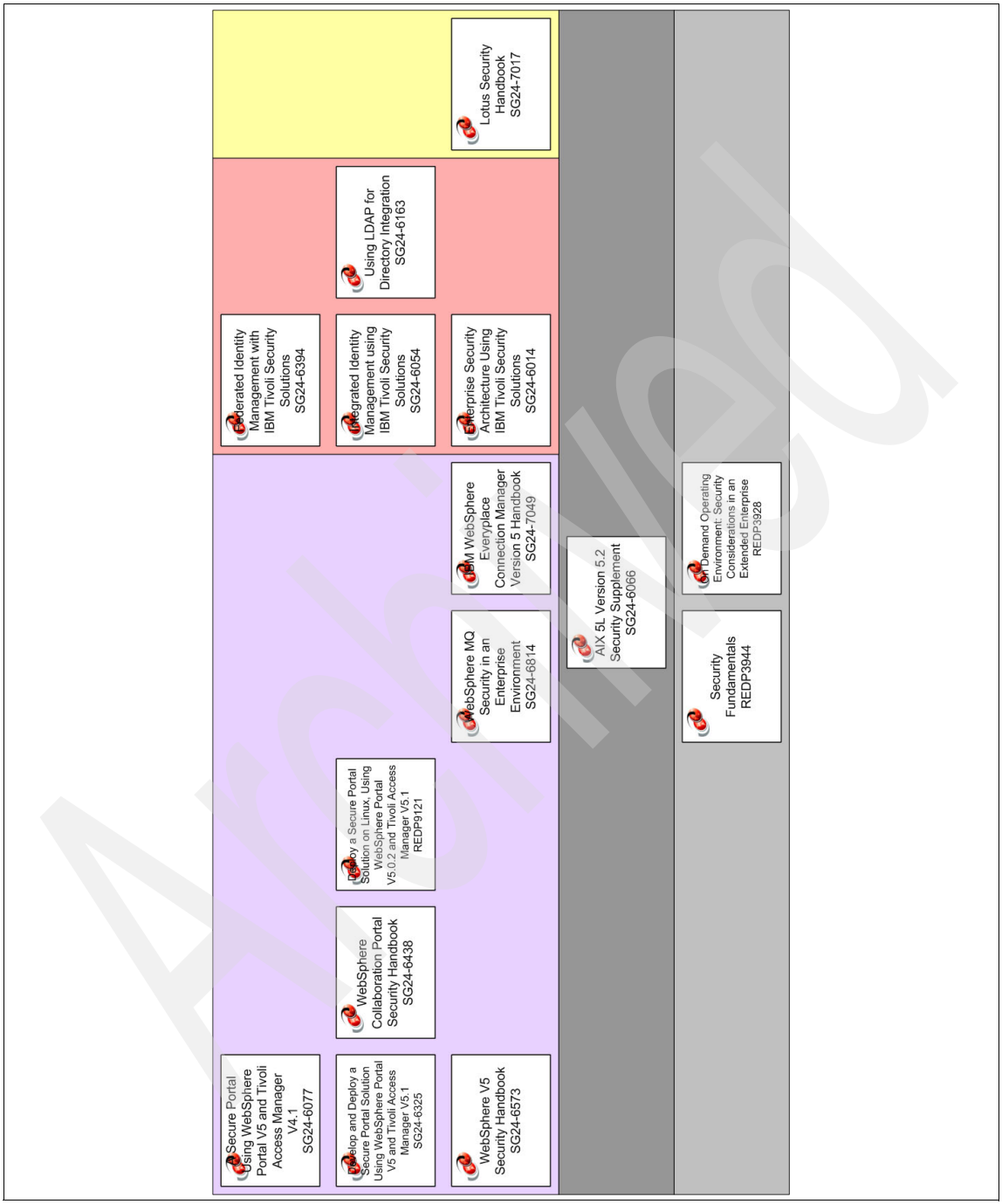


Figure 3-12 Security-related redbooks

Online resources

These Web sites are also relevant as further information sources:

- MIT's Kerberos Web site

  http://web.mit.edu/kerberos/www

- Sun's JACC Web site

  http://java.sun.com/j2ee/javaacc/index.html

- Sun's kinit description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/kinit.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/kinit.html

- Sun's JCE Web site

  http://java.sun.com/products/jce

- Sun's ktab description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/ktab.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/ktab.html

- Sun's jarsigner description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/jarsigner.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/jarsigner.html

- Sun's policytool description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/policytool.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/policytool.html

- Sun's keytool description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/solaris/keytool.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/keytool.html

- Sun's klist description

  http://java.sun.com/j2se/1.4.2/docs/tooldocs/linux/klist.html
  http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/klist.html

- Sun's security Web site

  http://java.sun.com/security/index.jsp

- Sun's Cetificate Path API Web site

  http://java.sun.com/j2se/1.4.2/docs/guide/security/certpath/CertPathPro

- IBM Java 2 security

  http://www-106.ibm.com/developerworks/java/jdk/security/142

- Sun's JSSE Web site

  http://java.sun.com/products/jsse

- Sun's JAAS Web site

  http://java.sun.com/products/jaas
- Sun's JGSS Web site

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jgss/tutorials/index
- IETF RFC 2853 Web site

  http://www.ietf.org/rfc/rfc2853.txt
- Microsoft SPNEGO Web site

  http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsecu

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

WebSphere Security Fundamentals

Redpaper

IT security fundamentals

Supporting security components for WebSphere

Security basics for J2SE, J2EE, and WebSphere

This IBM Redpaper is an overview of IT security for WebSphere products. Its purpose is to provide basic information about IT security and specific details about WebSphere security. This can be used as a standalone paper with information for basic security, or considered the introduction for WebSphere V6 Security and future security-related Redbooks.

Chapter 1, “Security fundamentals” discusses basic IT security without reference to any product.

Chapter 2, “Supporting security components for WebSphere” starts introducing IT components related to security without specifying product names.

Chapter 3, “Security fundamentals for J2SE, J2EE and WebSphere” goes into detail about J2EE and IBM WebSphere security. It provides an overview of security-related technologies and functions.

If you are not familiar with IT security in the Java 2 and WebSphere environments, this paper should be a good start.

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