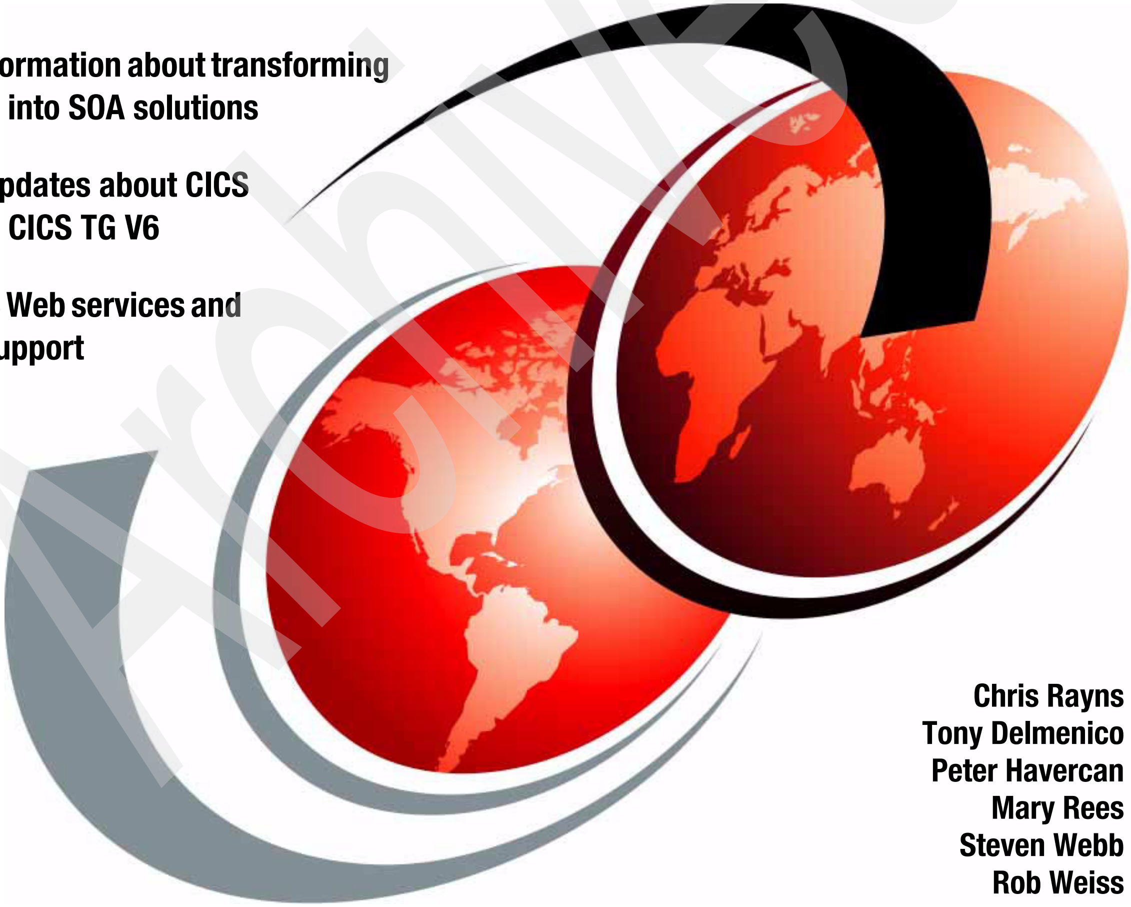
Securing Access to CICS Within an SOA



- Provides information about transforming CICS assets into SOA solutions
- Furnishes updates about CICS TS V3.1 and CICS TG V6
- Covers CICS Web services and CICS Web support

Chris Rayns
Tony Delmenico
Peter Havercan
Mary Rees
Steven Webb
Rob Weiss

Redbooks

ibm.com/redbooks

International Technical Support Organization

Securing Access to CICS Within an SOA

December 2006

SG24-5756-01

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

Second Edition (December 2006)

This edition applies to Version 3, Release 1 of CICS Transaction Server.

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 illustrate 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.

Trademarks

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

AIX®
CICS®
CICS/ESA®
CICSPlex®
DB2®
Distributed Relational Database Architecture™
DRDA®
eServer™
HiperSockets™
ibm.com®
IBM®
IMS™
Language Environment®
MVS™
OS/390®
Parallel Sysplex®
POWER™
RACF®
RDN™
Redbooks™
Redbooks (logo) ™
S/390®
SupportPac™
System z™
System z9™
Tivoli®
WebSphere®
z/Architecture™
z/OS®
zSeries®
z/VM®
z/VSE™
z9™

The following terms are trademarks of other companies:

SAP, and SAP logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries.

Enterprise JavaBeans, EJB, Java, JavaBeans, J2EE, Solaris, Sun, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries 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, or service names may be trademarks or service marks of others.

Preface

With the emergence of service-oriented architecture (SOA), the options for accessing the existing IBM® Customer Information Control System (CICS®) assets have become more varied than ever. With this variety comes the complexity of securing these assets. This IBM Redbook is intended for IT architects who are involved in the process of selecting, planning, and designing a secure SOA solution that makes use of CICS assets.

This book introduces SOA and the options available for transforming CICS assets into SOA solutions. It then discusses the principles of security, followed by the different security technologies.

The book then reviews each technology individually, discussing the security options that are available, the security architectures such as basic authentication, firewalls, and the use of Secure Sockets Layer (SSL) and public key infrastructure (PKI).

The team that wrote this IBM Redbook

This IBM Redbook was produced by a team of specialists from around the world working at the International Technical Support Organization (ITSO), Poughkeepsie Center.

Chris Rayns is an IT Specialist and Project Leader at the ITSO, Poughkeepsie Center in New York. Chris specializes in security and writes extensively about all areas of S/390® security. Before joining the ITSO, he worked in IBM Global Services in the United Kingdom (UK) as a CICS IT Specialist.

Tony Delmenico is a Senior Technical Specialist with IBM Global Services, Australia. He has 20 years of experience in IBM mainframes, working for customers and IBM. He has worked in a variety of technical support roles, but predominantly in CICS. He holds a Bachelor of Business in Information Technology degree from the Edith Cowan University.

Peter Havercan is a CICS Developer in the United Kingdom. He has over 20 years of experience in developing CICS. He has worked with IBM for 21 years. He holds a Bachelor of Science degree in Mathematics from the Imperial College, University of London, and a Master of Science degree in Physics from the University of Waterloo, Ontario. His areas of expertise include CICS Security and CICS Web Support, especially CICS support for SSL. He frequently gives presentations on these topics at CICS Technical Conferences.

Mary Rees is a CICS Knowledge Engineer in the United States. She has 16 years of experience in the CICS Level 2 and Knowledge Engineering field. She holds a degree in Chemical Engineering from the Michigan Technological University. Her areas of expertise include connectivity between CICS regions, terminal control, and security. Mary currently publishes Technotes to the CICS support page in order to assist in faster problem resolution.

Steven Webb is a CICS Level 2 Software Engineer in the United States. He has 17 years of experience in Level 2, with the last 10 years being with CICS. His primary areas of focus are CICS Web support, CICS Web Services, and Simple Object Access Protocol (SOAP). He holds a Bachelor of Science degree in Computer Science from the Michigan Technological University.

Rob Weiss is a z/Security Consultant and z/Software IT Architect in the United States. He has 37+ years of experience in IT, with a focus on Mainframe Security. He holds a degree in Mathematics from the Concord University, Athens, West Virginia. He has worked with Resource Access Control Facility (RACF®) since RACF's introduction. Rob has been involved in security consulting for commercial and governmental installations in several countries.

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

Richard Conway
Robert Haimowitz
ITSO, Poughkeepsie Center

Phil Wakelin
IBM Hursley (CICS Strategy and Planning)

Fraser Bohm
IBM Hursley (CICS Development Software Engineer)

The team that wrote the IBM Redbook *Implementing CICS*, SG24-7206, Nigel Williams, Grant Ward Able, Paolo Chieregatti, Robert Herman, Tommy Joergensen, Luis Aused Lopez, and Steve Wall.

Become a published author

Join us for a two-week 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 will have the opportunity to team with IBM technical professionals, business partners, and clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will 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!

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

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

  `ibm.com/redbooks`

- Send your comments in an e-mail to:

  `redbooks@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Summary of changes

This section describes the technical changes made in this edition of the IBM Redbook and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of Changes
for SG24-5756-01
for Securing Access to CICS Within an SOA
as created or updated in December 12, 2006.

December 2006, Second Edition

This revision reflects the addition, deletion, or modification of the new and changed information described below.

New information

- Chapter 1, “Introduction to SOA and CICS” on page 3
- Chapter 4, “CICS Web services” on page 121
- Chapter 7, “WebSphere MQ” on page 217

Changed information

- Chapter 2, “Security concepts” on page 23, Updated for z/OS® 1.8
- Chapter 3, “Security technologies” on page 93, Updated for z/OS 1.8
- Chapter 5, “CICS Web support” on page 157, Updated for CICS TS 3.1
- Chapter 6, “CICS Transaction Gateway” on page 185, Updated for CICS TG V6

Part 1

Security and CICS SOA access

Part 1 introduces service-oriented architecture (SOA) and its relationship to the CICS Transaction Server (TS). It also introduces the different styles that you can use when integrating the existing CICS assets into an SOA solution. It then describes the security concepts and the different security technologies.

1

Introduction to SOA and CICS

This chapter introduces the concepts of service-oriented architecture (SOA) and how they apply to IBM System z™, including a discussion about the business and IT benefits of SOA and an overview of the Web services technologies. This chapter then describes how you can transform the existing CICS assets to play a role in the SOA solutions.

This chapter comprises the following sections:

- 1.1, “SOA on System z” on page 4
- 1.2, “Transforming CICS assets into SOA solutions” on page 15
- 1.3, “Interaction between CICS and other core WebSphere SOA products” on page 21

1.1 SOA on System z

SOA is an architecture that organizations and their IT departments are adopting. But what is it? This section provides an overview of the SOA, the benefits it offers, and its positioning on System z.

1.1.1 Understanding SOA and the reasons for adopting it

This section discusses the growing use of SOA and attempts to explain what it means and how organizations can benefit by adopting it.

An introduction to SOA

SOA is an evolution of best practices and technologies that have preceded it. It combines the advantage of developments in Internet-based technology and interoperability standards to offer unrivalled business and IT benefits. SOA is defined in numerous ways, some more clearer than the others, but all of them lending themselves to the concept of loosely coupled business services that are provided in an interoperable and technology friendly manner.

SOA is an integration architecture approach that is based on the concept of *services*. The business and infrastructure functions that are required to build distributed systems are provided as services that individually or collectively deliver application functions to either user applications or to other services.

> **Note:** SOA is, as its name implies, an *architecture* that allows you to encapsulate business logic and separate it from application logic. It is *not* a formal specification. To create an SOA *implementation*, use a technology such as Web services or Service Component Architecture (SCA) in order to make the architecture a reality.

Going further, the definition of SOA can be viewed from the following *perspectives*:

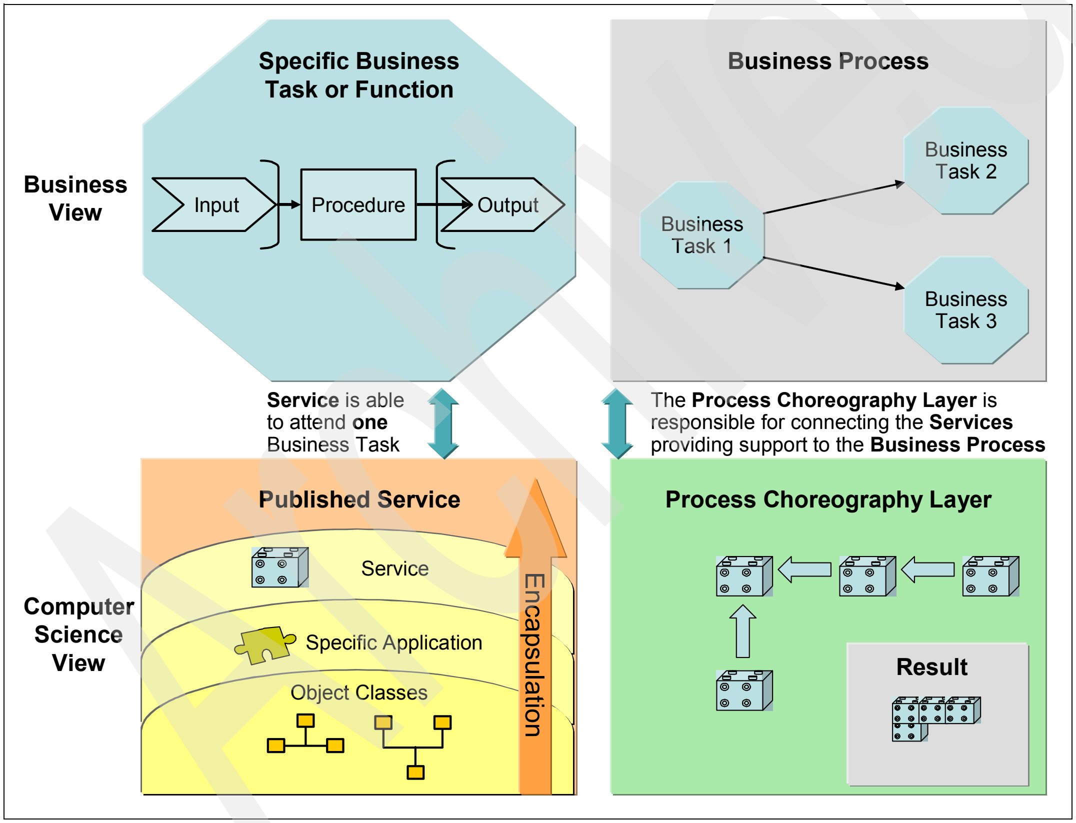
- A set of *business-aligned IT services* that support an organization's business goals and objectives
- A set of *architectural principles* that address characteristics such as modularity, loose coupling, and separation of functions
- An *architectural style* that requires a service provider, a service consumer, and a service description

- A set of services that can be combined and choreographed to produce *composite enterprise-scale services*
- A *programming model* that comes with standards, tools, methods, and technologies, such as Web services

By adopting an SOA approach and implementing it by using supporting technologies, companies can achieve the following:

- Build flexible systems that implement changing business processes quickly
- Make extensive use of reusable components

Figure 1-1 shows how services are invoked to support a particular business task or business process.



Business View
Specific Business Task or Function
Input
Procedure
Output
Business Process
Business Task 1
Business Task 2
Business Task 3
Service is able to attend one Business Task
The Process Choreography Layer is responsible for connecting the Services providing support to the Business Process
Computer Science View
Published Service
Service
Specific Application
Object Classes
Encapsulation
Process Choreography Layer
Result

Figure 1-1 Mapping services with business tasks or functions

Basic components of an SOA

At the most basic level, an SOA consists of the following three components (shown in Figure 1-2):

- Service provider
- Service requester
- Service registry

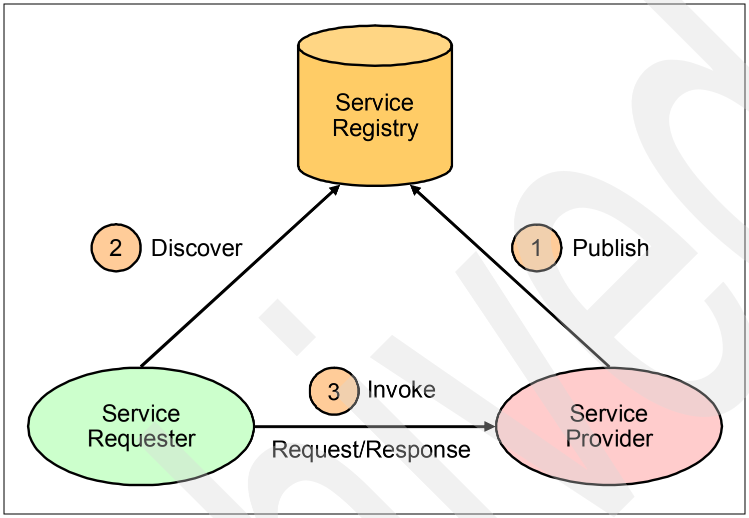


Figure 1-2 SOA components and operations

Service provider

The service provider creates a service, and in some cases, publishes its interface and access information to a service registry.

Each provider must decide which services to expose, evaluate tradeoffs between security and easy availability, determine how to price the services or figure out how to exploit the value of the services if they are free. The provider must also decide about the category the service must be listed to, and the sort of trading partner agreements required to use the service.

Service registry

The service registry is responsible for making the service interface and implementation access the information that is available to the service providers.

The implementers of a service registry must consider the scope within which the registry will be implemented. There are public service registries that are available over the Internet to an unrestricted audience, and private service registries that are only accessible to users within a company-wide intranet.

The service provider locates (discovers) entries in the service registry and then binds to the service provider to invoke the defined service.

Each of these components can also function as one of the two other components. For instance, if a service provider requires additional information that it can acquire only from another service, it behaves as a service requester.

Defining a service

SOA is an architectural approach to defining integration architectures that are based on the concept of services. A *service* can be described as a function that can be offered or provided to a requester. This function can be an atomic business function or a part of a collection of business functions that are wired together to form a process.

There are other additional aspects to a service that must also be considered when defining a service within an SOA. The most commonly agreed-on aspects are:

- Services encapsulate a reusable business function
- Services are defined by explicit, implementation-independent interfaces
- Services are invoked through communication protocols that stress location transparency and interoperability

Ideally, a service must be reusable and be accessible by more than one requesting application in the architecture. It is, therefore, important to get the service description and reusability correct, for example, a service that offers a calculation such as a home insurance quote can be requested by multiple requesters inside the enterprise and by third parties, as long as the interfaces of the component that offers the service are defined clearly.

Services can be invoked independently by either external or internal service requesters to process simple functions or can be chained together to form more complex functionalities in order to devise a new functionality quickly.

Clearly-defined interfaces

The interface for SOA must encapsulate only those aspects of process and behavior that are used in the interaction between the service requester and the service provider. An explicit interface definition or *contract* binds a service requester with the service provider. The interface must specify only the mutual behavior that is required for the interaction and nothing about the actual implementation of the requester or the provider.

This arrangement means that those system aspects, where the requester and the provider are hosted (their platforms), are independent of the interaction and are free to change. This allows for flexible improvements to the underlying IT infrastructure.

Communication protocols that stress location transparency

SOA does *not* specify that a consumer must have any specific protocol to access a service. A key principle in SOA is that a service is not defined by the communication protocol that it uses, but in a protocol-independent manner that allows different protocols to be used to access the same service. Ideally, a service must be defined only once, through a service interface, and must have many implementations with different access protocols. This type of definition helps increase the reusability of any service definition.

1.1.2 The business and IT benefits of SOA

This section provides a concise view of the business and IT benefits an organization can enjoy by adopting SOA.

Business benefits

Organizations will always seek out innovative methods to gain a competitive advantage. SOA allows the typically heterogeneous IT environment of an enterprise to be agile and responsive to fast-changing business conditions.

Following are some of the business advantages that SOA offers:

- The concept of components and reuse allows organizations to increase the speed with which they implement new products and services. By introducing new processes and data, changing the existing reusable elements, or recombining them quickly, technical support and provisioning of new products and services in the marketplace is enabled.
- The increased abstraction of business processes from implementation and run time concerns and constraints mean that there are fewer technical inhibitors that slow down progress and change.
- The modularity and reuse of components means that services are highly optimized to business requirements.
- The ability to *extract more from what is already there* means that organizations are able to introduce new capabilities that bring business advantages, for example, applications that were earlier siloed, can now work together behind the scenes and help shorten human-based processes and tasks.

- The ability to make available repeatable and reusable services across the enterprise means less duplication of functions and therefore, reduced instances of duplicated data such as client details. The ability to improve the quality of service and retain customers increases with more accurate information.

IT benefits

Because SOA is an approach that specifically aligns IT capabilities to business drivers and requirements, the distinction between what is an IT benefit as opposed to a business benefit, becomes somewhat blurred. Nevertheless, the IT benefits that an organization can realize by implementing SOA are included in the following list:

- The adoption of open standards and component-based development brings about long-term reductions in development costs and ongoing maintenance.
- The sharing of services and improved consistency reduces the duplication of siloed IT functions and consequently, the consolidation of hardware and software is made possible, thereby reducing costs.
- The revival of core applications through SOA capabilities reduces the requirement to replace such systems, thereby minimizing risk, disruption, and replacement costs.

According to Gartner, following are the benefits provided by SOA to IT:

“SOA will shift the focus from tools and packaged suites to modular offerings from multiple vendors that can be assembled and combined by a systems integrator. By 2008, SOA will provide the basis for 80 percent of development projects. By 2008, simple object database access plus service-oriented business applications (SOBAs) will enable Type A organizations to increase code reuse by more than 100 percent. The distinction between software integrators and vendors will blur because packaged applications will be broken up and delivered as SOBAs. In 2006, more than 60 percent of the $527 billion IT professional services market will be based on the exploitation of Web services standards and technology.”

Gartner also states: “SOA shifts developer focus from software to business functions, thereby transforming the installed software from an inhibitor to a facilitator of rapid business change.”

For more information, refer to *Positions on the Five Hottest IT Topics and Trends in 2005*, which is available on the Web at:

http://www.gartner.com

1.1.3 Web services

Web services are fast becoming the standard for basic SOA implementation. Web services take advantage of the existing open-standard Web technologies such as Extensible Markup Language (XML), Uniform Resource Locator (URL), and Hypertext Transfer Protocol (HTTP), and are themselves a set of standards that facilitate open system-to-system communication.

By adhering to Web services, standards applications that are based invariably on differentiating platforms and technologies, can cooperate through well-defined interfaces. Web services follow the SOA philosophy of loose coupling between service requesters and service providers. Figure 1-3 illustrates how loose coupling is maintained within the Web services interaction model.

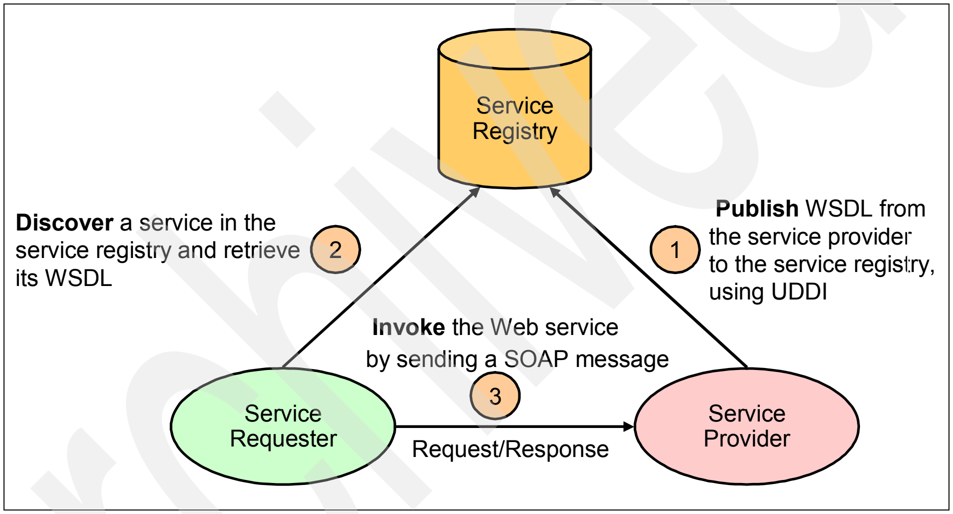

Service Registry
Discover a service in the service registry and retrieve its WSDL
2
Publish WSDL from the service provider to the service registry, using UDDI
1
Invoke the Web service by sending a SOAP message
3
Service Requester
Request/Response
Service Provider

Figure 1-3 Web services invocation model

The interaction shown in Figure 1-3 works as follows:

1. The service provider publishes the Web Services Description Language (WSDL) data that defines its interface and location to a service registry such as an Universal Description, Discovery, and Integration (UDDI) service registry.
2. The service requester contacts the service registry to obtain a reference to a service provider.
3. The service requester, having obtained the location of the service provider, makes calls on the service provider by sending a SOAP-formatted message.

Basic Web services support provides three simple usage models:

- One-way usage scenario

  A Web services message is sent from a requester to a provider, and no response message is expected

- Synchronous request or response usage scenario

  A Web services message is sent from a requester to a provider, and a response message is expected

- Basic callback usage scenario

  A Web service message is sent from a requester to a provider using the 2-way invocation model, but the response is treated only as an acknowledgement of a request having been received. The provider then responds by using a Web service callback to the requester.

Other Web service standards are built upon these basic standards and invocation models to provide higher level functions and qualities of service.

1.1.4 System z and the reason why it is appropriate for SOA

This section provides an overview of System z and the benefits to be gained from implementing SOA capabilities within an IBM z/OS environment.

System z

IBM eServer™ System z9™ (formerly IBM eServer zSeries®) is built on more than 40 years of industry leadership in mainframes. It uses a modular multibook design that supports one to four books per server. Multiple features such as redundant input/output (I/O) interconnections (RII) help avoid unplanned interruptions and outages. By increasing secure transaction throughput, System z9 can improve responsiveness when strengthening security through enhanced encryption and hashing algorithms.

System z contains specialized engines such as IBM System z9 Application Assist Processor (zAAP), IBM System z9 Integrated Information Processor (zIIP), IBM System z Integrated Facility for Linux® (IFL), and Internal Coupling Facility (ICF), which can all be used for your advantage. The virtualization and intelligent management features of IBM System z9 109 help reduce management complexity and facilitate a more efficient use of system resources.

z/OS

System z mainframes are supported by a multitude of operating systems such as z/OS, IBM z/OS.e, IBM z/VSE™, IBM z/VM®, Transaction Processing Facility (TPF), and Linux on System z. The flagship operating system of this group is z/OS. With its roots in MVS™ and IBM OS/390®, z/OS is the flagship mainframe operating system based on the IBM 64-bit z/Architecture™. It is designed to deliver a high quality of service for enterprise transactions and data, making it appropriate for larger enterprises.

Some highlights of z/OS V1.7 include the z/OS Workload Manager that helps balance resources, and Intelligent Resource Director (IRD), which extends the Workload Manager and makes it possible to manage resources dynamically across multiple logical partitions. z/OS Parallel Sysplex® technology allows you to balance workloads across multiple servers (up to 32) and is designed to provide near continuous availability.

The reason behind having an SOA framework on z/OS

In a sense, the mainframe environment has always lent itself to the concept of SOA because it regards *all* the resources within as providing services. Resources meant specifically for SOA are those that provide SOA capabilities such as the Enterprise Service Bus (ESB), process management engines, and supporting components such as a base Java™ 2, Enterprise Edition (J2EE™) application server and databases.

To offer the power of System z for SOA, IBM has developed specific z/OS versions of its SOA product suite that is built on IBM WebSphere® Application Server V6 for z/OS. The WebSphere Process Server and WebSphere Enterprise Service Bus are z/OS-enabled, as are supporting components such as IBM DB2® for z/OS V8. This offers a clean and contained architecture within a z/OS environment, with the architecture based on open and interoperability standards. Additionally, products such as the CICS Transaction Server have added features to support SOA technologies such as Web services, and can integrate with the WebSphere Application Server for z/OS-based products.

Following are the advantages of using System z and z/OS for SOA:

- Quality of service
- Core system transaction capabilities for SOA
- Cost of ownership

Quality of service

A framework that incorporates SOA capabilities exploits well-proven System z features such as high scalability, availability, reliability, and security. System z clustering is provided through Parallel Sysplex technology and workload management by z/OS Workload Manager (zWLM) to offer the following features:

- Less than five minutes downtime per year
- 99.999% availability at the application level

System z has built on four decades of development and collaboration to offer unparalleled security in both its hardware and z/OS operating system. In addition, the introduction of virtualization for z/OS helps decouple actual physical resources from users and services, thus bringing an additional layer of protection. For more details about security on System z, refer to:

`http://www.ibm.com/servers/eserver/zseries/security/features.html`

Core system transaction capabilities for SOA

The source of most services that service requesters call upon is usually core systems such as CICS and IBM Information Management System (IMS™) transactions. These core systems themselves can function as requesters and providers of services. The positioning of these systems within a System z environment means that performance is enhanced because of less network traffic and, in the case of z/OS, the HiperSocket technology leveraged. To facilitate connections to CICS and IMS for an SOA architecture, the CICS Transaction Gateway and the IMS SOAP Gateway V9.1 are offered.

Web services can be developed with IBM WebSphere Developer for zSeries tooling in order to generate Web services artifacts easily.

Cost of ownership

When the demand for computer usage increases every year, organizations introduce new boxes, systems, and applications in their widely heterogeneous and distributed IT environments. Managing these distributed environments can lead to the following hidden costs:

- Increased complexity
- Spiralling resource costs
- Increased downtime costs
- Suboptimized use of resources
- Licensing costs

The Wall Street Journal provides an interesting view:

“Distributed server farms today generate as much as 3,800 watts per square foot, compared to 250 watts per square foot in 1992, with thousands of dollars of cooling capacity needed for each server. Assuming 1,000 distributed servers are producing 400 watts each, the electricity bill could hit more than USD 35,000 per month alone. By comparison, a single mainframe z9 generates 312 watts per square foot – one tenth the amount.”

The centralized architecture of the mainframe has always helped avoid such issues, but initial purchase costs and operating costs were high. Recent developments in new technology for System z help reduce the total operating cost as follows:

- Virtualization

  Virtualization, which allows a single server or platform to support hundreds of concurrent applications and share data and hardware resources across heterogeneous environments, was invented by mainframes more than 35 years ago. Today, this is highly advantageous for enterprises that are looking for ways to simplify their IT infrastructures and reduce complexity and costs.

- zAAP

  To help lower costs, IBM introduced separate processing engines to tackle a collection of mainframe workload types. These engines can free your mainframe CPU for other tasks and lower the related capacity charges. The zAAP engine, released in 2005, reduces costs by processing Java-based application workloads.

- System z Integrated Information Processor (zIIP) engine

  DB2 works in concert with z/OS to tackle workloads that originate on distributed platforms (through IBM Distributed Relational Database Architecture™ (DRDA®) via TCP/IP) and access DB2 data running on the mainframe. Together, DB2 and zIIP help improve resource optimization for eligible workloads, including those from SAP® or other enterprise resource planning (ERP) applications, along with customer resource management (CRM) and business intelligence initiatives. A zIIP engine can be added as a one-time cost. It can then process up to 40% of such tasks with no additional software or capacity charges.

1.2 Transforming CICS assets into SOA solutions

Existing application assets running on a CICS Transaction Server can be utilized in SOA solutions in a number of ways. This section discusses the transformation strategies and describes how different types of CICS assets can be transformed.

1.2.1 Transformation strategies

The IBM SOA Reference Architecture shown in Figure 1-4 is a technical framework for enterprise transformation that enables software to be delivered as reusable, shareable services. This architecture provides the ability to bridge disparate systems spread across your entire enterprise. Because its components are modular, you can start small and grow your implementation to cover your evolving integration requirements, both internally and externally.

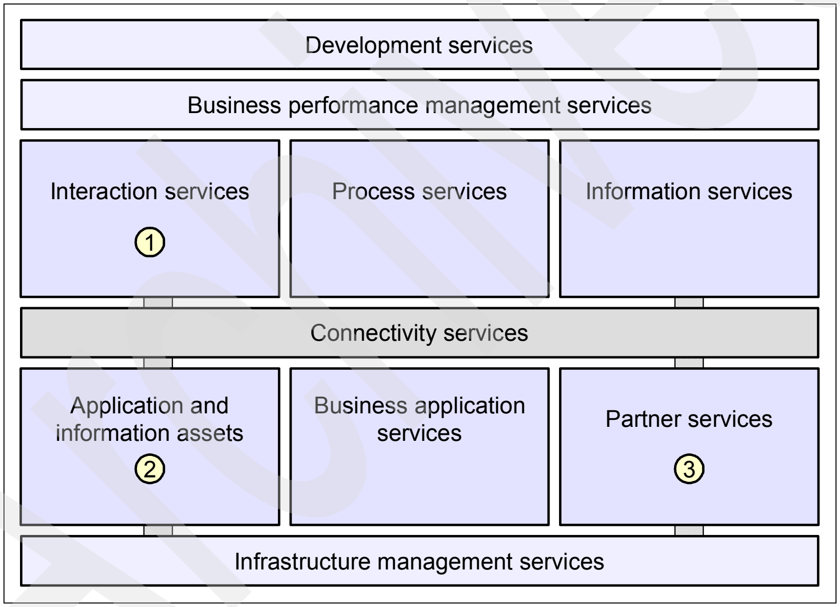

Development services
Business performance management services
Interaction services
1
Process services
Information services
Connectivity services
Application and information assets
2
Business application services
Partner services
3
Infrastructure management services

Figure 1-4 IBM SOA Reference Architecture

This section examines three components shown in Figure 1-4 in closer detail:

- User interface modernization

  This style (denoted by “1” in Figure 1-4) transforms the user experience. It aims to reach new customers even as it helps improve productivity and reduce costs. This style of transformation also helps reduce the training costs and increase the overall user satisfaction. This method is the most accessible because it requires the lowest level of investment. You can achieve a rapid return on investment (ROI) through improved user interfaces such as a modern interface design, and enhanced productivity with optimized interaction patterns.

- Application integration

  This style (denoted by “2” in Figure 1-4) transforms application connectivity. It aims to extend the existing applications beyond their original designs to support integrated business processes, helping reduce errors and development costs. You can turn the existing applications into reusable services that can be accessed by a new set of users or be reused to create new front-end business functions. The underlying principle that you can reuse the existing applications with little or no change, offers a lower-risk approach than a replacement strategy, which involves rewriting applications.

- Service orientation

  This style (denoted by “3” in Figure 1-4) transforms the application architecture to provide greater responsiveness to business partners and customers. It involves some re-engineering of the original application. Undoubtedly, this method requires higher investment of resources and time, but gives you the capability to create components from the existing applications, which are more flexible and can be configured for use in new applications. This reuse of business logic is called *componentization* and may result in significant cost savings when compared to the costs involved in developing new application code.

1.2.2 The CICS assets that can be transformed

Over the past 35 years, developers have created two major types of CICS applications or assets:

- CICS COMMAREA programs

  These programs receive requests and send responses through an area of memory called the COMMunications AREA (COMMAREA). CICS programs can be written in COBOL, PL/I, C, C++, Assembler, or Java. In general, CICS COMMAREA programs are similar to subroutines, in that, they are unaware of how they were invoked. They are often stateless, with CICS, on behalf of the program, automatically managing the transactional scope and security context, which are typically inherited from a caller and a transaction definition.

- CICS terminal-oriented programs

  These programs are sometimes known as *3270 programs* because they are designed to be invoked directly from an IBM 3270 Display Station or similar buffered terminal device. Invocation usually corresponds to a single interaction in a user dialog, starting with the receipt of a message from the terminal and ending with the transmission of a reply message to the same device. Input data from the terminal device is carried in a datastream, which the application acquires through a RECEIVE command. After processing, an output datastream is transmitted back to the terminal device through a SEND command. Terminal-oriented programs must be capable of analyzing device-specific input data streams and building the output data streams that are to be transmitted to the terminal.

CICS also provides a service known as *basic mapping support* (BMS), which simplifies application programming for terminals such as the 3270 Display Station. This enables the programmer to define a static layout for each screen to be displayed, with identified fields for dynamic content acquired through a RECEIVE MAP command. This in turn causes BMS to analyze the datastream and to return record-formatted data to the application. Similarly, the application presents output data in record format using a SEND MAP command, which causes BMS to build an output datastream for the terminal. BMS is widely used because it frees the application programmer from requiring knowledge about device specifics, and enables applications to be device-independent to some degree.

A *pseudo-conversational* model is normally associated with terminal-oriented transactions. A pseudo-conversational sequence of transactions contains a series of transactions that look to the user, such as a single conversational transaction involving several screens of input. However, each transaction in the sequence is, in fact, a single transaction that handles one input, sends back the response, and then terminates.

1.2.3 Access to COMMAREA programs

The best practice in CICS application design for a number of years now has been to separate the key elements of the application, the following in particular:

- Client adapt or presentation logic
- Integration logic
- Business logic
- Data access logic

Figure 1-5 shows a transaction made up of these separate components. A COMMAREA interface, which includes channels with CICS Transaction Server (TS) V3.1, is used to pass data between the components.

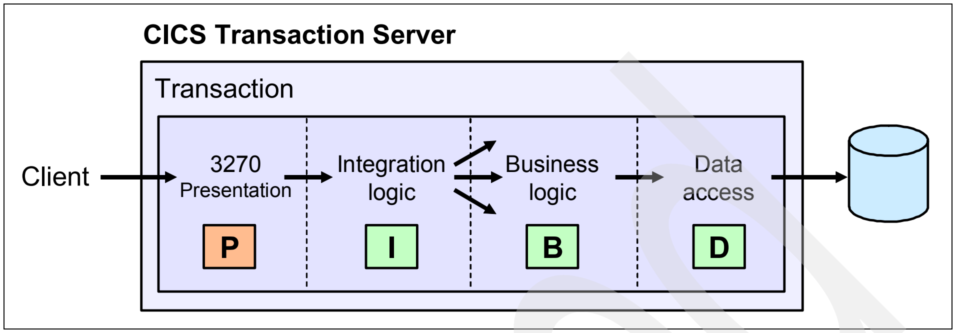


Figure 1-5 Separating the key application elements

This separation provides a framework that enables the reuse of business logic and data access logic programs as subroutines within a larger application and reuse with alternative implementations of presentation logic, for example, a Web service, a Web browser, or a 3270 device.

CICS COMMAREA programs can be relatively easily enabled for access from a variety of different client applications running on a wide range of platforms. Typical clients include:

- Web service requester
- Java servlet or Enterprise JavaBeans™ (EJB™) running on a J2EE application server
- An application running on a Microsoft® .NET environment
- Web browser
- Messaging middleware

In most cases, connections from a client use a combination of the following:

- Internal adapters
- External connectors
- Standard IP-based protocols

An adapter is simply a program that accepts a request and converts the data from an external format to the internal format used by the CICS business logic program. Figure 1-6 shows how a terminal-oriented program and a Web service requester can access the same CICS applications. An adapter can, for example, convert a SOAP message to a COMMAREA format. The transport mechanism used to invoke the adapter may be synchronous or asynchronous.

An internal adapter is run-time code, possibly generated by a tool that converts from one request format to another, such as converting SOAP over HTTP to a COMMAREA. You can implement the adapter in any language that is supported by CICS and make it independent of the specific protocol that is used.

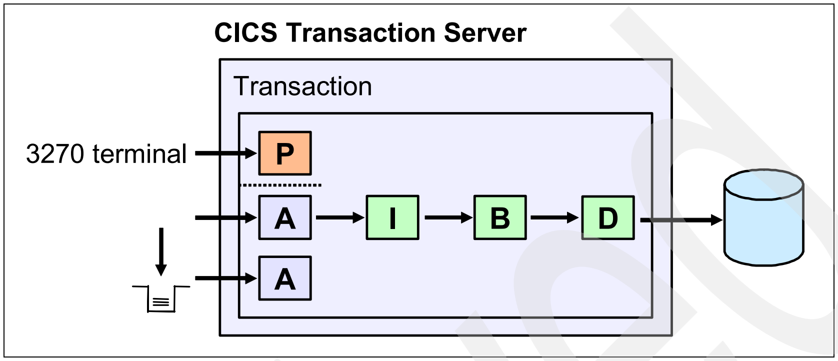


Figure 1-6 Access options provided by CICS facilitate reuse of existing business logic

An external connector provides a remote call interface and implements a private protocol to invoke an application running under the CICS Transaction Server. You must also use an external adapter to convert data from its external format to the COMMAREA format that is used by your programs in the CICS Transaction Server. The most well-known example of an external connector is IBM CICS Transaction Gateway, which implements the Common Connector Interface specified by the J2EE Connector Architecture (JCA), and is used with adapters implemented as Java beans.

Along with these techniques, you can create a standard IP-based adapter that uses a specific transport such as IBM WebSphere MQ, HTTP, and TCP/IP sockets. This approach may be the only available option that supports some types of clients. This method also permits greater flexibility in the functionality that can be implemented. However, this flexibility must be balanced against additional development effort, and a loss of generality and reuse because you can use the adapter only with a specific transport protocol.

Your preferred architectural approach is a key decision because of its effect on the cost of developing the solution and its long-term return on investment (ROI). However, business factors such as existing development processes and the availability of skills may be as significant as the technical factors influencing this decision. It is important to recognize the fact that there is no single correct answer that is suitable for all the solutions.

1.2.4 Access to terminal-oriented programs

There are many programs that do not have such a clear separation of concerns as COMMAREA programs, combining a presentation logic ("P" in Figure 1-7) and business logic ("B" in Figure 1-7) into a single program, for which there is only a 3270 interface.

IBM CICS TS V3.1 provides a Link3270 bridge function that neatly addresses this problem. The client uses the Link3270 bridge to run 3270 transactions by linking to the DFHL3270 program and passing a COMMAREA that includes the transaction identifier and the data to be passed to the application. The response contains the 3270 screen data reply. If the target application uses basic mapping support (BMS), this information is presented in the form of an application data structure (ADS), which is another name for the symbolic map that is generated by the BMS macros used to define the mapping of the 3270 screen. No changes are required for the existing application code. Knowledge about the 3270 data streams is usually not required. As a result, the Link3270 bridge provides a programmatic interface for an important class of terminal-oriented programs, enabling them to be reused without resorting to less efficient and more fragile screen scraping.

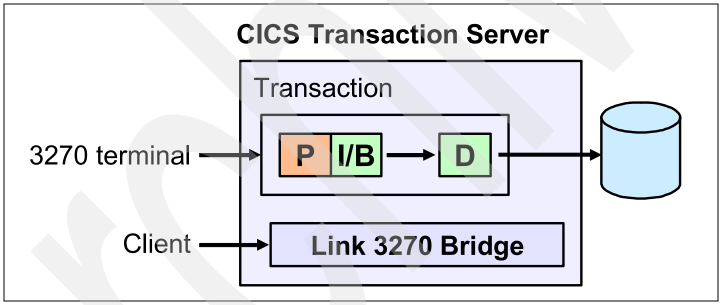


Figure 1-7 Access options provided by CICS facilitate reuse of existing terminal-oriented programs

Historically, many 3270 transactions were written as pseudo-conversations, consisting of a number of terminal-oriented programs that run in a defined sequence. Each program in a pseudo-conversation displays data to a user and then terminates, leaving only a small amount of state data to be picked up by the next program in the sequence, which is initiated by the next input data received from the user's terminal. The Link3270 bridge is able to fully reuse these pseudo-conversational transactions.

CICS programs are typically grouped into application suites or components for performing a common set of business actions. Identifying the CICS programs that provide flexible public interfaces and understanding these interfaces is the first key step for reuse. The next step is to decide on the best access options to support your solution.

1.3 Interaction between CICS and other core WebSphere SOA products

CICS TS provides the features that are necessary to build complete SOA solutions. Service requesters can use these features to gain access to the CICS assets. However, CICS TS does *not* have to be used in isolation to build SOA solutions.

The following products are also important to building SOA implementations, and can be used to interact with CICS TS assets:

- WebSphere Application Server

  It hosts J2EE enterprise applications such as EJBs and Web services. Enterprise applications deployed to the WebSphere Application Server can interact with CICS assets using a variety of techniques described in this IBM Redbook.

  For more information about the WebSphere Application Server, refer to:

  http://www.ibm.com/software/webservers/appserv/was/

- WebSphere Enterprise Service Bus

  This mediates SOAP, Java Message Service (JMS), and Internet Inter-ORB Protocol (IIOP) messages, as they travel between service requesters and service providers. The mediations provide content-based routing to pick the service provider that must be used, the protocol and message transformation, and so on. SOA interactions to or from the CICS TS can be mediated in the WebSphere Enterprise Service Bus.

  For more information about the WebSphere Enterprise Service Bus, refer to:

  http://www.ibm.com/software/integration/wsesb/

- WebSphere Process Server

  This manages business processes that incorporate calls to automated activities such as Web services and manual activities such as those completed by a person. Calls to CICS assets can be incorporated into business processes running on WebSphere Process Server.

  For more information about the WebSphere Process Server, refer to:

  http://www.ibm.com/software/integration/wps/

Note: Each of the three products listed can run on the z/OS and the distributed platforms.

2

Security concepts

This chapter lays the groundwork for a discussion of the security functions and security issues in each of the products discussed in this IBM Redbook. This chapter also describes the requirements for an IAA (Identification, Authentication, and Authorization) solution..

2.1 The importance of security

Effective identity management, authorization, and access control capabilities are the essential elements of a comprehensive enterprise security program. These security services are necessary to control the access to resources and sensitive information within the IT environment. The introduction of new and pending legislation relating to the handling and access of sensitive information increases the importance of these vital functions. Requirements such as reduced sign-on, role-based access control, centralized authorization policies, timely account management, and synchronized directories can be fulfilled, with the solution approach outlined.

In order to manage core business processes, companies must give their users, who are as likely to be customers or business partners as employees, access to corporate information and applications through a wide-ranging network. Organizations require a unified and centralized approach for making authorization decisions instead of relying on custom access control services for each server, application, or environment. The necessity for a unified and centralized approach is also driven by the requirement to be responsive to changes in user population. New hires must be given appropriate access as quickly as possible. Users, whose responsibilities change due to promotion or changes in job responsibilities, must have their authorizations changed quickly to reflect the change in responsibilities. Most importantly, employees who leave the company must have all their authorizations revoked on a real-time basis. Furthermore, companies require an adaptable solution that can scale up according to the demand. By providing highly available and centralized authorization services, companies can better manage and secure their business-critical distributed information and ensure their ability to meet the time-to-market, flexibility, and scalability requirements that today's on demand world requires.

The increased use of Web technologies has changed the business landscape over the past few years. Web technologies are being used to deliver information and to provide access to client accounts, manage sales, and conduct transactions with both business partners and customers. These are just a few examples of the manner in which the Internet has transformed business practices. With this increased connectivity and subsequent access to corporate information, it becomes critical to properly identify the entities that request information. Requestors must only be able to access information or applications that they are authorized to access.

2.2 Identification, Authentication, and Authorization

An IAA (Identification, Authentication, and Authorization) solution must provide services that support the following business processes:

- User registration and administration
  - Creation and deletion
    - Provisioning
    - Administration
  - Suspend and resume
  - Entitlements management
- User login and logout
  - Authentication
  - Password failure lock-out
  - Application of idle “time-out” policy

For a more comprehensive strategy, your solution must include the following:

- User resource requests
  - Authorization of requests
- User self-care
  - Self-registration
  - Password change
  - Password reset

These solutions are for supporting IT architectural initiatives in order to reduce the complexity and cost by implementing reusable functions. This may require some of the processes supported by IAA to be defined at an enterprise level.

These items are *not* discussed in this book. However, they must be addressed by your IT Security Officer.

2.3 IAA requirements

The IAA recommendations outlined in this section provide implementation-independent information for identity management, authentication, and authorization services.

2.3.1 IT environment and application architecture requirements

Following are the IT environment and application architecture requirements:

- IAA must be designed to ensure that it can be extended to cover the authentication and authorization of access to internal and external Web services.
- IAA must be designed to ensure that it can be extended to cover the authentication and authorization of access to a single service that is delivered internally and externally.
- IAA must integrate with the internal desktop SOA when it is defined.
- IAA must integrate with internal and extended global enterprise directory strategies as they mature for the evolving, extended global enterprise.
- IAA must support the mapping of Web service identities into associated identities for established security databases such as Resource Access Control Facility (RACF).
- IAA must reflect the component-based architecture, shielding applications from changes in technology and promoting reuse.
- For custom software development, IAA must support the Java 2, Enterprise Edition (J2EE) application environment and the associated standards such as Enterprise JavaBeans (EJB), Java Authentication and Authorization Service (JAAS), and so on.
- Where possible, implement IAA services in such a way that decisions are made “on behalf of” components, rather than by them.
- Provide standardized application programming interfaces (APIs) for access to IAA services.
- Provide support for publishing directory changes to, and accept changes from a global meta directory.

2.3.2 IAA component security

In addition to the system and network security requirements imposed by the IT infrastructure, IAA must comply with the following security requirements:

- All IAA management and administrative actions must have the capability to be logged to a secure audit trail, and retained according to the audit logging policies.
- All IAA communication between the IAA components must use secure channels, providing authenticated and encrypted communication between the components.

- All IAA authentication information such as passwords and keys must be stored in an encrypted form to protect it from unauthorized disclosure. The z/OS Security Manager, that is, RACF, must contain and manage all the password information.
- All the IAA keys used for the encryption of stored and in-transit information must be managed in line with the relevant key management standards. Keys for the Secure Sockets Layer (SSL) are managed by a z/OS service in recent releases of z/OS.
- All IAA management and administrative interfaces must be protected by the authentication and authorization mechanisms that comply with the relevant logical access control standards.
- All IAA authentication actions performed on behalf of applications must be capable of being logged to a secure audit trail, and retained according to local and legislatively required audit logging policies.
- All unsuccessful IAA authorization actions performed on behalf of applications must be logged to a secure audit trail and retained according to local and legislatively required audit logging policies.

2.3.3 Application-managed security

To be defined on an application-by-application basis for each application or group that is integrated into the IAA strategy, the consideration must be to externalize all the application security management to the z/OS Security Manager (RACF). This allows the centralization of security management.

Application-specific authentication, authorization, and access control can cause inconsistencies in the level of protection for information assets, service interruptions, and data compromises. The same confidential information can, for example, be accessed by applications with either strong or poorly implemented security mechanisms.

Organizations incur significant costs over developing, testing, and maintaining custom-coded security, including additional ongoing costs to maintain the infrastructure. Each time a new initiative is deployed in this manner, an additional unique user account that has to be managed is created. This can result in increased programming costs, spiralling mainframe exploitation, and delayed deployment of new initiatives.

Authentication and access control may require additional security mechanisms that the existing application's security model does not support. Some applications may not, for example, support certificate-based authentication or may not provide the required level of access control granularity that an external authentication and authorization component supports.

2.3.4 Availability and performance requirements

The IAA solution must be architected to provide resilience, availability, scalability, and performance in line with the design and projections for the overall system environment.

Components of the IAA must be designed and implemented to provide the same levels of availability as the other equivalent components in the existing environment.

The online components of IAA must be designed and implemented to meet the requirements of the business, including the following:

- No single point of failure
- Automatic failover and switchback
- Automatic switching to the backup site within the required time periods

The management components can be designed to meet lower availability requirements.

2.3.5 Performance and scalability requirements

It must be possible to scale the capacity of the proposed architecture to meet the future sizing requirements. It must also be possible to scale the IAA capacity by expanding the capacity of the existing servers and by adding additional servers to the IAA configuration.

IAA performance and capacity metrics must cover the following parameters:

- Concurrent user sessions
- Resource request rate
- User login rate
- Growth of user base before requiring additional resources to be deployed

2.3.6 Identity management structures

This architecture provides a consistent IAA infrastructure to deliver effective and efficient security across the infrastructure. Following are the core security components of the recommended solution:

- To centralize user account provisioning by automating the management and provisioning of user identities for the infrastructure, including applications, user registries, and operating systems
- To provide password synchronization by reducing the number of unique passwords that a user requires to maintain and simplify authentication, and decrease the requirement of password resets

- Distributed administration through the delegation of user administration from a central authoritative resource. An installation can substantially improve the security administration and complexity of credential life cycle management. Often, this can be automated through feeds from the human resources data changes. This requires the use of role-based security administration.

2.4 The role of cryptography

A complete security policy puts the necessary mechanisms in place to achieve the following objectives:

- Identification

  This is the ability to assign an identity to the entity accessing the system. Typically, identity is used to control access to resources. Depending on the security model in which the identification is performed, the identity can be called a *user ID,* a *UID*, or a *principal*.

- Authentication

  This is the process of validating the identity claimed by the accessing entity. Authentication is performed by verifying the authentication information provided by the claimed identity. The authentication information is generally referred to as the accessor's *credentials*. A credential can be the accessor's name and password or can be a *token* provided by a trusted party, for example, a Kerberos ticket or an X.509 certificate.

  Authentication is a must when you want to provide different rights to access resources such as files and databases to different requesting identities.

  Note: Authentication is usually one of the earliest steps in a request workflow. When authenticated, an identity can be *asserted* to the downstream process steps, meaning these steps trust the upstream steps to have already authenticated the identity successfully.

- Authorization

  Authorization is the process of checking whether an identity that has already been authenticated must be provided access to a resource that it is requesting. A typical implementation of authorization is to pass a *security context* that contains the identity that has been authenticated, to the access control mechanism.

- Integrity

  Integrity ensures that transmitted or stored information has *not* been altered in an unauthorized or accidental manner. Typically, it is a mechanism to verify whether what is received over a network is the same as what was sent.

- Confidentiality

  Confidentiality ensures that an unauthorized party does not obtain the meaning of the transferred or stored data. Typically, confidentiality is achieved by encrypting the data.

- Auditing

  With auditing, you capture and record security-related events such as a user signing in to a system or out of a system so that you can analyze the events if a security breach occurs at a future date.

- Nonrepudiation

  Nonrepudiation means that a data sender and a data receiver are able to provide legal proof to a third party that the sender did send the information, and the receiver received the identical information. Neither side is *able to deny*.

In computer security, cryptography provides the following processes:

- *Encrypting* converts plaintext (data in normal, readable form) into ciphertext, which conceals the meaning of the data to any unauthorized recipient. Encrypting is also called enciphering.

  Most cryptographic systems combine the following two elements:

  - An algorithm that specifies the mathematical steps required to encrypt the data
  - A cryptographic key (a string of numbers or characters) or keys. The algorithm uses the key to select one relationship between plaintext and ciphertext out of the many possible relationships the algorithm provides. The selected relationship determines the composition of the algorithm's result.

- *Decrypting* converts ciphertext back into plaintext. Decrypting is also called deciphering.
- *Hashing* uses a one-way (irreversible) calculation to condense a long message into a compact bit string called a message digest.

- Generating a *digital signature* involves encrypting a message digest with a private key to create the electronic equivalent of a handwritten signature. You can use a digital signature to verify the identity of the person who signed and to ensure that nothing has been altered in the signed document from the time it was signed.

This chapter shows you how to use cryptography in order to achieve authentication, data integrity, confidentiality, and nonrepudiation.

2.5 Secret key or symmetric cryptography

In secret key cryptography, the sender and the receiver of a message know and use the same secret key. The sender uses the secret key to encrypt the message and the receiver uses the same secret key to decrypt the message (see Figure 2-1). Secret key cryptography is also known as symmetric cryptography.

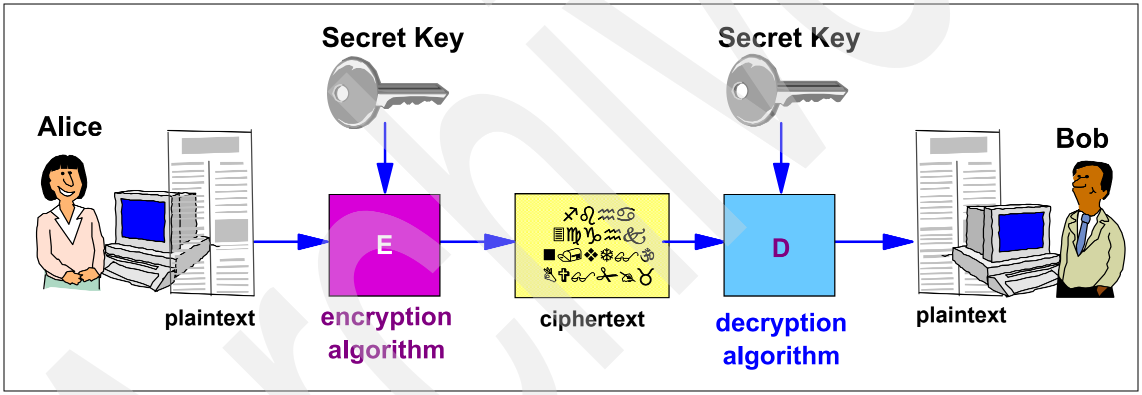


Figure 2-1 Secret key or symmetric cryptography

The main challenge of secret key cryptography is getting the sender and the receiver to agree on the secret key without anyone else finding out. If the sender and the receiver are in separate physical locations, they must trust a courier, a phone system, or some other transmission medium to prevent the disclosure of the secret key. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all the messages that are encrypted using that key.

Block ciphers

A block cipher is a type of secret key encryption algorithm that transforms a *fixed length* block of plaintext data into a block of ciphertext data of the same length. This transformation takes place under the action of a user-provided secret key. Decryption is performed by applying the reverse transformation to the ciphertext block using the same secret key. The fixed length is called the block size. The common block sizes are 64 bits and 128 bits.

Iterated block ciphers

Iterated block ciphers encrypt a plaintext block with the help of a process involving several rounds. In each round, the same transformation, also known as a round function, is applied to the data using a subkey. The set of subkeys is usually derived from the user-provided secret key by a special function. The set of subkeys is called the key schedule. The number of rounds in an iterated cipher depends on the desired security level and the consequent trade-off with performance. In most cases, an increased number of rounds improve the security offered by a block cipher.

2.5.1 Data Encryption Algorithm or Data Encryption Standard

The Data Encryption Algorithm (DEA) developed by IBM is an example of an iterated block cipher. IBM submitted the DEA to the National Bureau of Standards (NBS) during an NBS public solicitation for cryptographic algorithms to be used in a Federal Information Processing Standard (FIPS). In 1977, the NBS issued the FIPS Publication 46, *Data Encryption Standard (DES)*, which specified that the DEA be used within the United States Federal Government for the cryptographic protection of sensitive, but unclassified, computer data. As a result, the DEA is often called the DES.

The DES was reaffirmed in 1983, 1988, 1993, and 1999. As time passed, the NBS became the National Institute of Standards and Technology or NIST, a division of the US Department of Commerce.

The DES has a 64-bit block size. A DES key consists of 64 bits, of which 56 bits are randomly generated and used directly by the algorithm. The other 8 bits, which are not used by the algorithm, can be used for error detection. Following is the binary format of the key:

```
(B1,B2,...,B7,P1,B8,...,B14,P2,B15,...,B49,P7,B50,...,B56,P8)
```

Here, B1,B2,...B56 are the independent bits of a DES key and P1,P2,...P8 are reserved for parity bits computed on the preceding seven independent bits and set, so that the parity of the octet is odd. In other words, there is an odd number of “1” bits in the octet.

DES modes of operation

When a block cipher is used to encrypt a message of *arbitrary* length, techniques known as modes of operation are used for the block cipher. In December 1980, FIPS Publication 81, *DES Modes of Operation*, announced four modes of operation for DES:

- Electronic Codebook (ECB)
- Cipher Block Chaining (CBC)
- Cipher Feedback (CFB)
- Output Feedback (OFB)

The ECB and CBC modes of operation are described in the following sections.

Electronic Codebook

In ECB mode, the message M of arbitrary length is first divided into blocks m_i. Each block contains 64 bits, the block size of the DES algorithm. Each plaintext block m_i is used directly as the input block to the DES algorithm. The resultant output block is used directly as ciphertext (see Figure 2-2).

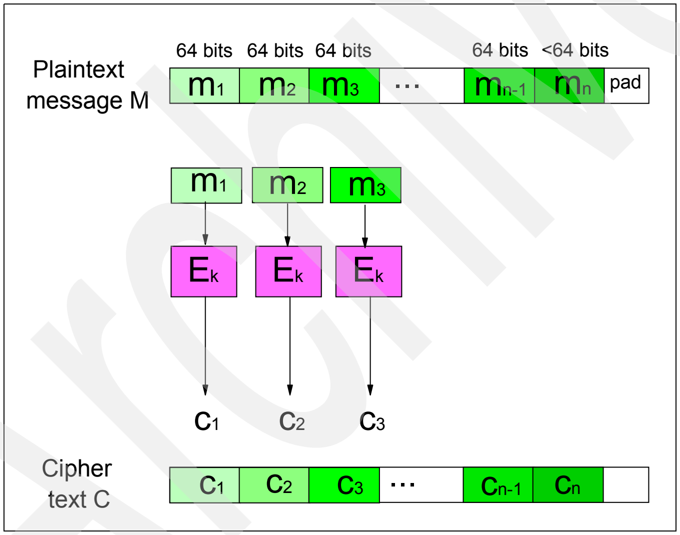


Figure 2-2 ECB mode of operation

The analogy to a codebook arises because the same plaintext block always produces the same ciphertext block for a given cryptographic key. Thus, a list or codebook of plaintext blocks and the corresponding ciphertext blocks can theoretically be constructed for any given key.

Because the ECB mode is a 64-bit block cipher, an ECB device must encrypt data in integral multiples of 64 bits. If a user has less than 64 bits to encrypt, the least significant bits of the unused portion of the input data block must be padded, for example, filled with random or pseudorandom bits prior to ECB encryption. The corresponding decrypting device must then discard these padding bits after the decryption of the ciphertext block.

Cipher Block Chaining

In practice, CBC is the most widely used mode of DES. In CBC, the message M of arbitrary length is first divided into blocks m_i. Each block contains 64 bits, the block size of the DES algorithm. Each plaintext block m_i is XORed (exclusive ORed) with the previous ciphertext block c_{i-1} and then encrypted. A 64-bit initialization vector c_0 is used as a "seed" for the process (see Figure 2-3).

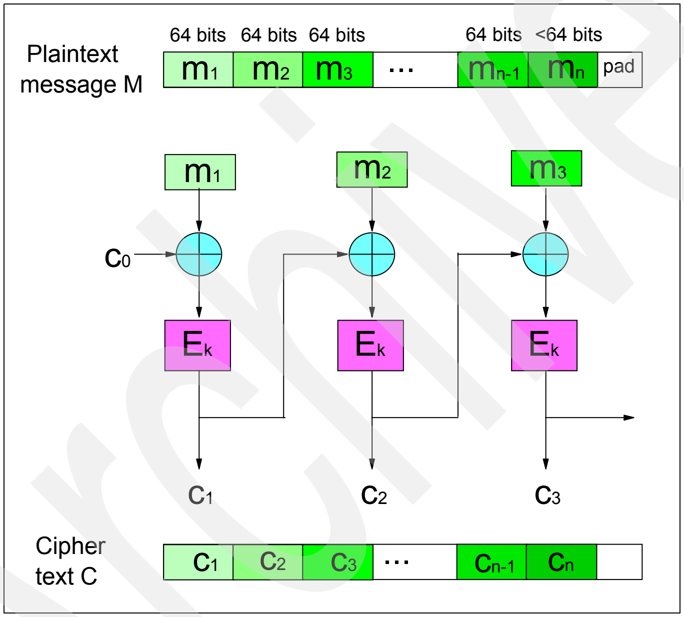


Figure 2-3 CBC mode of operation

Thus, the encryption of each block depends on the previous blocks, and the same 64-bit plaintext block can encrypt to different ciphertext blocks depending on its context in the overall message. XORing of the previous ciphertext block with the plaintext block conceals any patterns in the plaintext.

Partial data blocks (of less than 64 bits) require special handling. One of the methods of encrypting the final partial data block of a message is described here.

Use this method for applications where the length of the ciphertext is greater than the length of the plaintext. In this case, the final partial data block of a message is padded in the least significant bits positions with “0”s, “1”s, or pseudorandom bits. The decryptor must know when and to what extent padding has occurred. This can, for example, be accomplished explicitly by using a padding indicator or implicitly, by using constant length transactions.

The padding indicator depends on the data that is being encrypted. Following are the types of data:

- Binary

  If the data is purely binary, the partial data block must be left justified in the input block and the unused bits of the block set to the complement of the last data bit, that is, if the last data bit of the message is “0”, then “1”s are used as padding bits. If the last data bit is “1”, then “0”s are used. The input block is then encrypted.

  The resulting output block is the ciphertext. The ciphertext message must be marked as being padded, so that the decryptor can reverse the padding process, remove the padding bits, and produce the original plaintext. The decryptor scans the decrypted padded block and discards the least significant bits that are all identical.

- Bytes

  If the data consists of bytes, for example, 8-bit American Standard Code for Information Interchange (ASCII) characters, the padding indicator must be a character denoting the number of padding bytes, including itself, and must be placed in the least significant byte of the input block before encrypting. If, for example, there are five ASCII data characters in the final partial block of a message to be encrypted, then an ASCII “3” is put in the least significant byte of the input block (any pad characters may be used in the other two pad positions) before encryption. Again, the ciphertext message must be marked as being padded.

Status of the Data Encryption Standard

Because the speed of computers has increased significantly since 1977, it may now be possible to try every possible DES 56-bit key in turn until the correct key is identified. This technique of attempting to decipher a message is called exhaustive key search or brute force search. In fact, a DES cracking machine has been used to recover a DES key in 22 hours.

Therefore, the consensus of the cryptographic community is that DES is no longer secure. FIPS 46-3 reaffirmed DES usage as of October 1999, but permitted single DES only for established systems. FIPS 46-3 included a definition of triple DES (TDEA) which became “the FIPS approved symmetric encryption algorithm of choice”. On November 26, 2001, the NIST published FIPS 197 announcing the Advanced Encryption Standard (AES). The standard became effective on May 26, 2002. The NIST withdrew FIPS 46-3 on May 19, 2005.

2.5.2 Triple DES (TDEA)

For some time now, it has been common practice to protect information with triple DES instead of DES. This means that the input data is, in effect, encrypted three times. There are different ways of doing this. FIPS Pub 46-3 defines triple DES encryption with keys k_1, k_2, and k_3 as follows:

$$C = E_{k3}(\ D_{k2}(\ E_{k1}(M)\ )\ )$$

Here, $E_k(I)$ and $D_k(I)$ denote DES encryption and DES decryption respectively, of the input I with the key k (Figure 2-4).

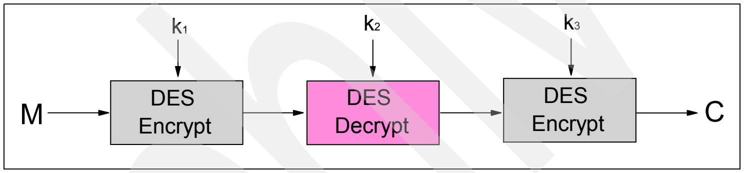


Figure 2-4 Triple DES-EDE

This mode of encryption is sometimes referred to as DES-EDE (encrypt, decrypt, encrypt). FIPS Pub 46-3 defines three keying options for DES-EDE:

- k_1, k_2, and k_3 are independent
- k_1 and k_2 are independent, but $k_3 = k_1$
- $k_1 = k_2 = k_3$

Another variant is DES-EEE, which consists of three consecutive encryptions.

Triple Data Encryption Algorithm modes of operation

Like all block ciphers, triple DES can be used in a variety of modes. The American National Standards Institute (ANSI) X9.52 standard *Triple Data Encryption Algorithm Modes of Operation* describes seven different modes:

- TDEA Electronic Codebook (TECB)
- TDEA Cipher Block Chaining (TCBC)
- TDEA Cipher Block Chaining - Interleaved (TCBC - I)

- TDEA Cipher Feedback (TCFB)
- TDEA Cipher Feedback - Pipelined (TCFB-P)
- TDEA Output Feedback (TOFB)
- TDEA Output Feedback - Interleaved (TOFB-I)

2.5.3 Advanced Encryption Standard

The Advanced Encryption Standard (AES) is another example of an iterated block cipher. The AES algorithm resulted from a multiyear evaluation process led by the NIST with submissions and reviews by an international community of cryptography experts. The Rijndael algorithm, invented by Joan Daemen and Vincent Rijmen, was selected as the standard. The NIST specified the AES in FIPS PUB 197 in November 2001.

The AES processes data blocks of 128 bits, in that, the input and the output for the AES algorithm each consists of a sequence of 128 bits (16 bytes or four words).

The cipher key for the AES algorithm is a sequence of 128, 192, or 256 bits. These different “flavors” of AES may be referred to as “AES-128”, “AES-192”, and “AES-256”. The number of words (Nk) in the key is thus 4, 6, or 8.

The number of rounds (Nr) to be performed during the execution of the algorithm depends on the key size. When Nk = 4, then Nr = 10. If Nk=6, then Nr=12, and when Nk=8, then Nr=14.

2.6 Public key or asymmetric cryptography

In public key cryptography, each person gets a pair of keys, one called the public key and the other called the private key. The public key is published and the private key is kept a secret. The necessity for the sender and the receiver to share the secret information is eliminated. All communications involve only public keys, and no private key is ever transmitted or shared. In this system, it is not essential to trust the security of some means of communication. The only requirement is that public keys be associated with their users in a trusted manner, for instance, in a trusted directory.

In public key cryptography:

- Data encrypted with a public key can only be decrypted with the corresponding private key. This guarantees data privacy for the receiver because only the receiver can decrypt the data. However, the receiver cannot be sure about who the sender is; it could be anybody.

- Data encrypted with a private key can only be decrypted with the corresponding public key. Anybody can decrypt the data, but the receiver knows who the sender is because the data can come only from one sender, the owner of the private key.

When Alice wishes to send a secret message to Bob, she looks up Bob's public key in a directory, uses it to encrypt the message, and sends it. Bob then uses his private key to decrypt the message and read it. No one listening in can decrypt the message. Anyone can send an encrypted message to Bob, but only Bob can read it because only Bob knows his private key (see Figure 2-5). Public key cryptography is also known as asymmetric cryptography.

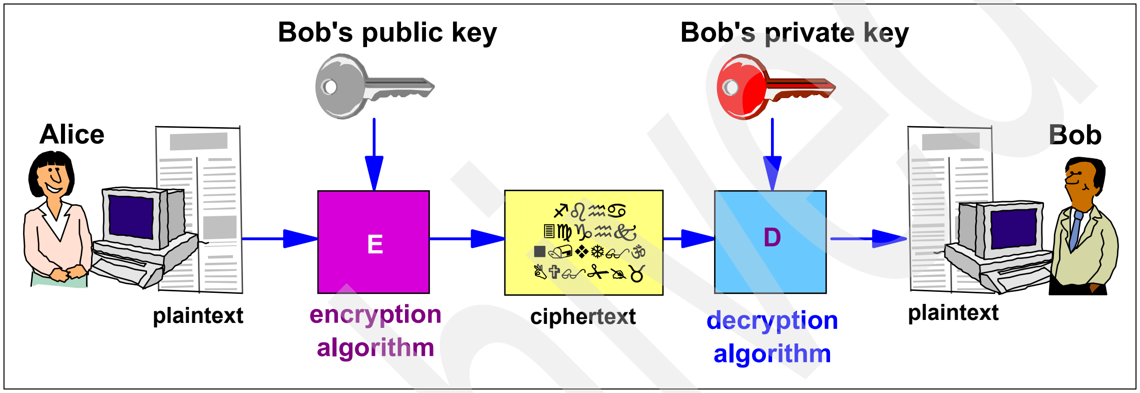


Figure 2-5 Public key or asymmetric cryptography

Notice how public key cryptography solves the problem of how to safely transmit a secret key. When Alice wants to send a secret key to Bob, she looks up Bob's public key in a directory, uses it to encrypt the secret key, and sends it. Bob then uses his private key to decrypt the secret key and read it. No one listening in can decrypt the secret key.

In a public key cryptosystem, the private key is always linked mathematically to the public key. Therefore, it is always possible to attack a public key system by deriving the private key from the public key. Typically, the defense against this is to make the problem of deriving the private key from the public key as difficult as possible. For instance, some public key cryptosystems are designed in such a way that deriving the private key from the public key requires the attacker to factor a large number. In such a situation, it is computationally not feasible to perform the derivation.

2.6.1 The Rivest-Shamir-Adleman algorithm

The Rivest-Shamir-Adleman (RSA) algorithm cryptosystem is a public key cryptosystem developed in 1977 by Ronald Rivest, Adi Shamir, and Leonard Adleman. The RSA algorithm is by far the most widely used public key cryptosystem in the world.

Before discussing how the RSA algorithm works, this section reviews the following definitions:

- Prime number

  This is any integer greater than 1 that is divisible only by 1 and itself. The first 12 primes are 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, and 37.

- Factor

  Given an integer n, any number that divides it is called a factor of n, for example, 7 is a factor of 91, because 91/7 is an integer.

- Factoring

  This is the breaking down of an integer into its prime factors. This is a hard problem, that is, a computationally-intensive problem or a problem that is computationally difficult to solve.

- Relatively prime

  Two integers are relatively prime if they have no common factors, for example, 14 and 25 are relatively prime, but 14 and 91 are not (7 is a common factor of 14 and 91).

- Congruent modulo n

  Given integers a, b, and n with n > 0, we say that a and b are congruent modulo n if a-b is divisible by n, that is, if (a-b)/n = i, an integer. Equivalently, a and b are congruent modulo n if there is an integer i, such that a-b = i x n, that is, such that a = b + (i x n). If a and b are congruent modulo n, we write a = b mod n, for example, 50 = 0 mod 5 because (50 - 0)/5 =10, an integer. But 50 is not congruent to 1 mod 5 because (50-1)/5 is not an integer. However, 51 is congruent to 1 mod 5.

  In arithmetic mod n, integers between 0 and n - 1 are used with normal addition, subtraction, multiplication, and exponentiation, except that after each operation, the result keeps only the remainder after dividing by n, for example, 5^6 mod 23 = 8 because 5^6=5 x 5 x 5 x 5 x 5 x 5 = 15,625 and 15,625 divided by 23 gives a quotient of 679 and a remainder of 8. Also, 9 + 7 = 1 mod 5 because 9 + 7 = 16 and 16 divided by 5 gives a quotient of 3 and a remainder of 1.

The RSA algorithm works as follows:

Take two large primes, p and q, and compute their product n = pq. Choose a number, e, less than n and relatively prime to (p-1)(q-1). Find another number d such that (ed - 1) is divisible by (p-1)(q-1). The public key is the pair (n, e) and the private key is (n, d). The factors p and q may be destroyed or kept with the private key.

RSA uses the following terminology:

- n is called the modulus
- e is called the public exponent
- d is called the private exponent

It is currently difficult to obtain the private exponent d from the public key (n, e). However, if you can factor n into p and q, you can obtain the private exponent d. Thus, the security of the RSA system is based on the assumption that factoring is difficult. The discovery of an easy method of factoring will "break" RSA.

Here is how the RSA system can be used for encryption. Let us suppose that Alice wants to send a message m to Bob. Alice creates the ciphertext c by exponentiating $c = m^e$ mod n, where e and n are Bob's public key. She sends c to Bob. To decrypt, Bob also exponentiates $m = c^d$ mod n. The relationship between e and d ensures that Bob correctly recovers m. Because only Bob knows d, and only Bob can decrypt this message.

In practice, however, the RSA system is often used together with a secret key cryptosystem such as DES. If Alice wants to send an encrypted message to Bob, she first encrypts the message with DES, using a randomly chosen DES key. She then looks up Bob's public key and uses it to encrypt the DES key. The DES-encrypted message and the RSA-encrypted DES key are sent to Bob. When he receives them, Bob decrypts the DES key with his private key and uses the DES key to decrypt the message. This combines the high speed of DES with the key management convenience of the RSA system.

Using a large key in the RSA cryptosystem

The size of a key in the RSA algorithm typically refers to the size of n. The two primes, p and q, which compose n must be of roughly equal length. This makes n harder to factor than if one of the primes is much smaller than the other. If you choose to use a 768-bit value for n, the primes must each have a length of approximately 384 bits.

The best size for n depends on your security requirements. The larger the value of n, the greater the security, but also the slower the RSA algorithm operations. Select a length for n based on the following considerations:

- The value of the protected data and how long it must be protected
- How powerful your potential threats may be

Key sizes of 512-bits no longer provide sufficient security for anything more than very short-term security requirements. RSA Laboratories currently recommends key sizes of 1024 bits for corporate use and 2048 bits for extremely valuable keys such as the root key pair used by a certifying authority. Less valuable information may well be encrypted using a 768-bit key. As such, a key is still beyond the reach of all known key-breaking algorithms. RSA Laboratories publishes recommended key lengths on a regular basis.

With regard to the slowdown caused by increasing the key size, doubling the length of n will, on an average, increase the time required for public key operations (encryption and signature verification) by a factor of four, and increase the time taken by private key operations (decrypting and signing) by a factor of eight. Key generation time increases by a factor of 16 when the length of n is doubled, but this is a relatively infrequent operation for most users.

2.7 Hash functions

A hash function H is a transformation that takes an input message m and returns a fixed-size string, which is called the hash value h. Using a mathematical notation for functions, this is expressed as h=H(m) (see Figure 2-6).

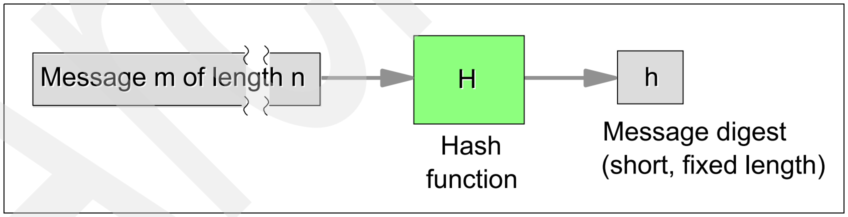


Figure 2-6 A hash function

When employed in cryptography, hash functions are usually chosen to have some additional properties. The basic requirements for a cryptographic hash function are as follows:

- The input can be of any length
- The output has a fixed length
- H(m) is relatively easy to compute for any given m
- H(m) is one-way

A hash function H is said to be one-way if it is hard to invert. This means that given a hash value h, it is computationally infeasible to find some input m, such that H(m)=h. An everyday example of a one-way function is mashing a potato. You can mash it easily, but after you have mashed it, it is rather difficult to reconstruct the original potato.

- H(m) is collision-free

  A collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y, such that H(x)=h and H(y)=h. That is, it is computationally infeasible to find any two messages that hash to the same value.

The hash value concisely represents the longer message or document from which it is computed. This value is called the *message digest*. A message digest can be thought of as a "digital fingerprint" of the larger document. It identifies the message much like an actual fingerprint identifies a person. Thus, a good cryptographic hash function ensures that it is difficult to:

- Recover the message from the message digest
- Construct a block of data M_2 that has the same message digest h as another given block, M_1

Use the hashing function to verify that the data has not been altered during transmission. The sender of the data calculates the message digest using the data itself and the hashing function. The sender then ensures that the message digest is transmitted *with integrity* to the intended receiver of the data. One way of doing this is to publish the message digest in a reliable source of public information. When the receiver gets the data, the receiver can generate the message digest and compare it with the original one. If the two are equal, the receiver accepts the data as genuine, and if they differ, the receiver assumes that the data is false.

In this example, the message digest must not be sent in the clear. Because the hash functions are well-known and no key is involved, a man-in-the-middle can not only forge the message, but also replace the message digest with that of the forged message. This makes it impossible for the receiver to detect forgery.

I. Damgard and R.C. Merkle greatly influenced the cryptographic hash function design by defining a hash function in terms of what is called a *compression function*. A compression function takes a fixed-length input and returns a shorter, fixed-length output. Given a compression function F, a hash function can be defined by repeated applications of the compression function F until the entire message is processed. In this process, a message of arbitrary length is broken into blocks (the length depends on the compression function) and "padded" (for security reasons), so that the size of the message is a multiple of the block size. The blocks are then processed sequentially, taking as input the result of the hash so far and the current message block, with the final output being the hash value for the message (see Figure 2-7).

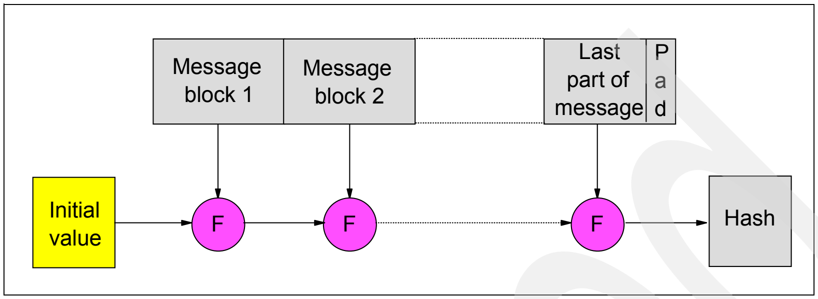


Figure 2-7 Iterative structure for hash functions

Following is a list of well-known hash functions:

- MD2 and MD5

  These were developed by Ronald Rivest of the Laboratory for Computer Science at the Massachusetts Institute of Technology (MIT). Both functions take a message of arbitrary length and produce a 128-bit message digest. MD2 was optimized for 8-bit machines, and MD5 was aimed at 32-bit machines. The description and source code for MD2 and MD5 can be found as Internet Request for Comments (RFCs) 1319 and 1321, respectively.

- SHA-1

  The Secure Hash Algorithm (SHA), the algorithm specified in the Secure Hash Standard (FIPS PUB 180), was developed by NIST. SHA-1 is a revision of SHA that was published in 1994. The revision corrected an unpublished flaw in SHA.

  The algorithm takes a message of less than 2^{64} bits in length and produces a 160-bit message digest. The algorithm is slightly slower than MD5, but the larger message digest makes it more secure against brute force collision and inversion attacks.

2.8 Message authentication codes

There are four types of message authentication codes (MACs):

- Unconditionally secure
- Stream cipher-based
- Block cipher-based
- Hash function-based

MACs based on cryptographic hash functions are known as Keyed Hashing Message Authentication Code (HMACs). These have two functionally distinct parameters, a message input and a secret key known only to the message originator and the intended receivers.

An HMAC function is used by the sender of the message to produce a value (the MAC) that is formed by condensing the secret key and the message input. The MAC is typically sent to the receiver of the message along with the message. The receiver computes the MAC on the received message, using the same key and HMAC function used by the sender, and compares the computed result with the received MAC. If the two values match, it means that the message has been correctly received, and the receiver can be assured that the sender is a member of the community of users who share the key (see Figure 2-8).

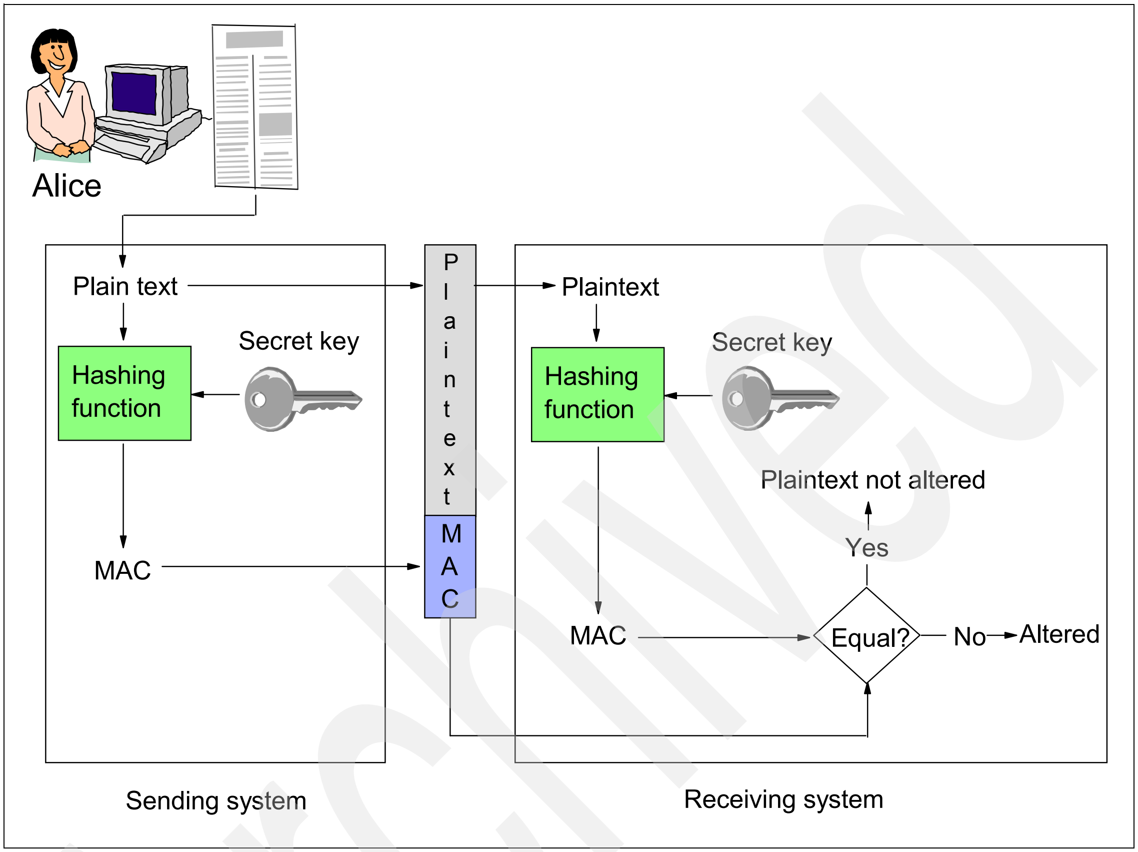


Figure 2-8 Keyed Hash Message Authentication Code

Note that because the receiver has the key that is used in the creation of the MAC, this process does *not* offer a guarantee of nonrepudiation because it is theoretically possible for the receiver to forge a message and claim it was sent by the sender.

2.9 Digital signatures

When public key cryptography is used to calculate a digital signature, the sender encrypts the message digest of the document with the sender's private key. Anybody who has access to the public key of the person who signed can verify the signature. Following is an example of the same:

1. If Alice wants to send a signed document or message to Bob, she applies a hash function to the message, creating a message digest. She then encrypts the message digest with her private key, thereby creating the digital signature. (Because the message digest is usually considerably shorter than the original message, Alice saves a considerable amount of time when she encrypts the message digest rather than the message itself).
2. Alice sends Bob the encrypted message digest (digital signature) and the message.
3. On receiving the message and the signature, Bob decrypts the signature with Alice's public key to recover the message digest. He then hashes the message with the same hash function that Alice used and compares the result with the message digest decrypted from the signature.

If they are exactly equal, it means that the signature has been verified successfully and he can be confident that the message did indeed come from Alice. If they are not equal, it means that either the message originated elsewhere or was altered after it was signed. Bob then rejects the message. Figure 2-9 shows this process.

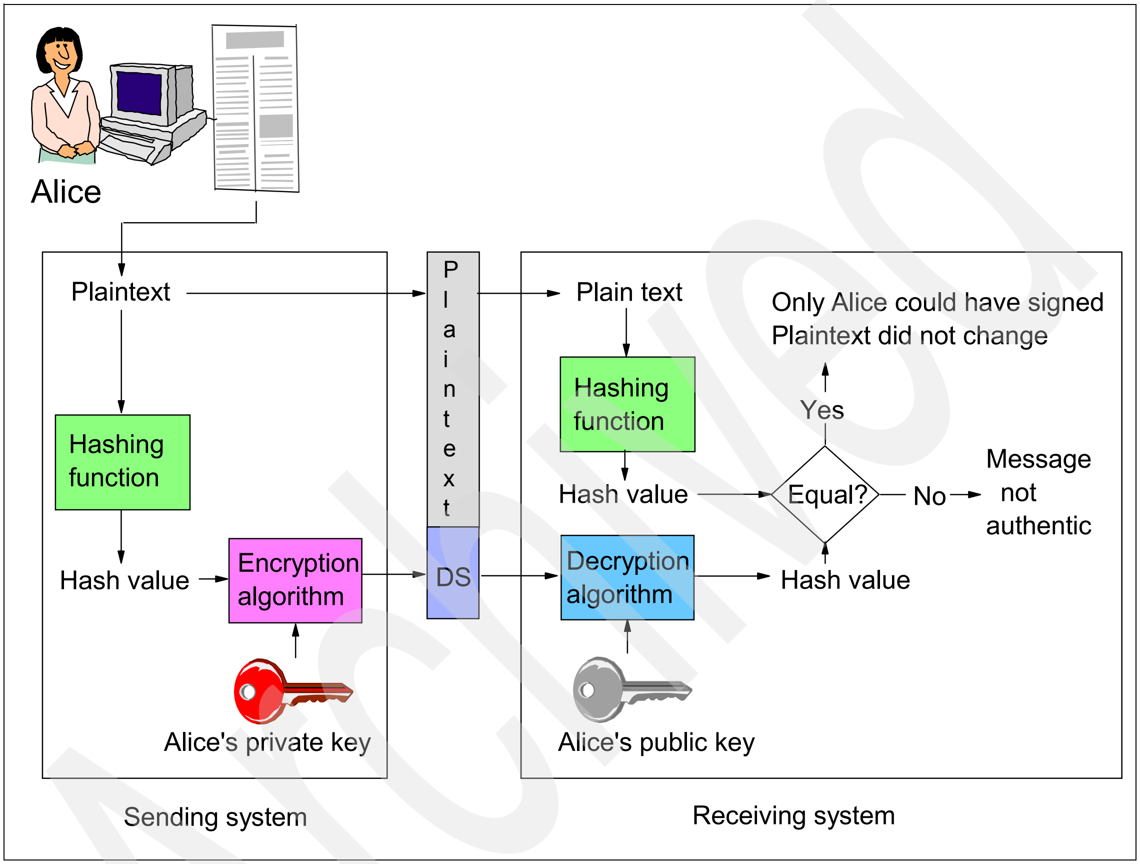


Figure 2-9 Creating and verifying a digital signature in a public key system

Note that the recipient of the signed data can use a digital signature to prove to a third party that the signature is, in fact, generated by the signatory. This is known as *nonrepudiation* because, at a later date, the signatory cannot repudiate the signature.

The following is a slightly different scenario based on the previously described scenario:

1. Alice may want to keep the contents of the document a secret. In such a situation, she may want to sign the document and then encrypt it using Bob's public key.
2. Bob will then have to decrypt the document using his private key and verify the signature on the recovered message by using Alice's public key.

   Figure 2-10 describes this process.

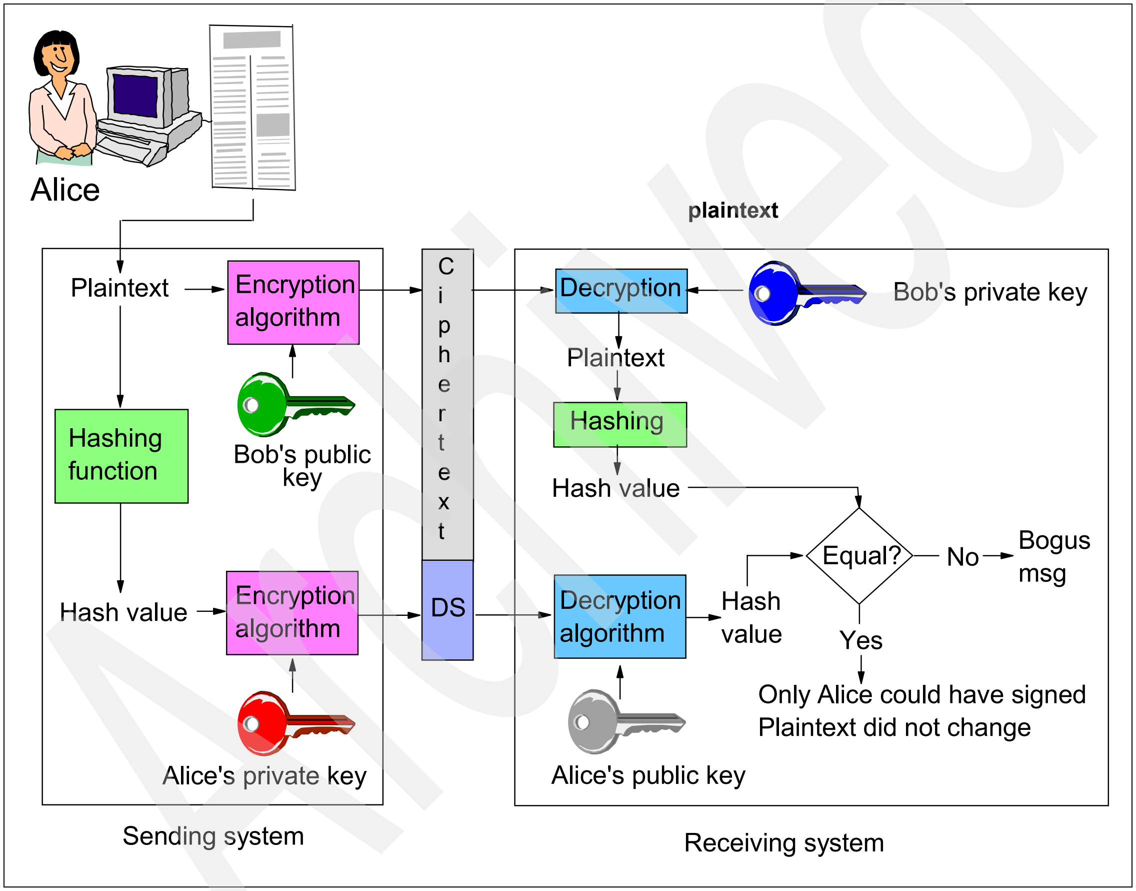


Figure 2-10 Creating and verifying a digital signature when encrypting the message

Alternately, if it is necessary for intermediary third parties to validate the integrity of the message without being able to decrypt its content, a message digest may be computed on the encrypted message, rather than on its plaintext form.

There is a potential problem with this type of digital signature. Alice not only signed the message she intended to sign, but she also signed all the other messages that happen to hash to the same message digest. When two messages hash to the same message digest, it is called a collision. The collision-free properties of hash functions are a necessary security requirement for most digital signature schemes.

In addition, someone may pretend to be Alice and sign the documents with a key pair the pretender claims is Alice's. To avoid scenarios such as this, there are digital documents called certificates that associate a person with a specific public key. For more information about digital certificates, refer to 2.10, "Public key digital certificates" on page 51.

2.9.1 Using the RSA algorithm for digital signatures

Use the RSA algorithm to compute a digital signature. If Alice wants to send a message m to Bob, she applies a hash function H to the message m, creating a message digest h=H(m). She then creates a digital signature s by exponentiating $s = h^d \bmod n$, where d and n are Alice's private keys. She sends m and s to Bob. To verify the signature, Bob exponentiates and checks whether the message digest h is recovered, that is, $h = s^e \bmod n$, where e and n are Alice's public keys.

In practice, the public exponent in the RSA algorithm is usually much smaller than the private exponent. This means that the verification of a signature is faster than signing. This is desirable because a message will be signed by an individual only once, but the signature is likely to be verified many times.

2.9.2 Using the Digital Signature Algorithm for digital signatures

The RSA system can be used for both encryption and digital signatures, but the Digital Signature Algorithm (DSA) can only be used to provide digital signatures. The NIST published the first version of the DSA in the *Digital Signature Standard (DSS)* FIPS PUB 186 in May 1994. The current version was published in FIPS PUB 186-2 in January 2000. In October 2001, Change Notice 1 amended FIPS PUB 186-2.

Digital Signature Algorithm parameters

The DSA makes use of the following parameters:

- p = A prime number, where $2^{1023} < p < 2^{1024}$

  Appendix 2 of FIPS PUB 186-2 specifies a method of generating p.

- q = A prime divisor of p-1, where $2^{159} < q < 2^{160}$

  Appendix 2 of FIPS PUB 186-2 also specifies a method of generating q.

- $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p-1$ such that $h^{(p-1)/q} \bmod p > 1$
- x = A randomly or pseudo-randomly generated integer with $0 < x < q$
- $y = g^x \bmod p$
- k = A randomly or pseudo-randomly generated integer with $0 < k < q$

  Appendix 3 of FIPS PUB 186-2 specifies a method of generating both x and k.

The integers p, q, and g can be public and can be common to a group of users. The integers x and y are a user's private and public keys, respectively. The parameters x and k are used only for signature generation, and must be kept a secret. The parameter k must be regenerated for each signature.

Digital Signature Algorithm signature generation

The signature of a message M is the pair of numbers r and s computed according to the following equations:

- $r = (g^k \bmod p) \bmod q$
- $s = (k^{-1}(\text{SHA-1}(M) + xr)) \bmod q$

Here, k^{-1} is the multiplicative inverse of k, mod q, that is, $(k^{-1}\ k) \bmod q = 1$ and $0 < k^{-1} < q$. The value of SHA-1(M) is a 160-bit string output by the Secure Hash Algorithm SHA-1. For use in computing s, this string must be converted to an integer.

Digital Signature Algorithm signature verification

Before verifying the signature in a signed message, p, q, and g plus, the sender's public key y and identity are made available to the verifier in an authenticated manner.

If M', r', and s' are the received versions of M, r, and s respectively, to verify the signature, the verifier first checks to see that $0 < r' < q$ and $0 < s' < q$. If either condition is violated, the signature must be rejected. If these two conditions are satisfied, the verifier computes the following:

- $w = (s')^{-1} \bmod q$
- $u1 = (\ (\ \text{SHA-1}(M')\ )w\ ) \bmod q$
- $u2 = (\ (r')w\ ) \bmod q$
- $v = (\ (\ g^{u1}\ y^{u2}) \bmod p\ ) \bmod q$

If v = r', the signature is verified and the verifier can be highly confident that the received message was sent by the party holding the secret key x corresponding to y.

If v does not equal r’, the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message must be considered as invalid.

2.9.3 Using the Elliptic Curve Digital Signature Algorithm for digital signatures

The ANSI X9.62 standard *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)* specifies a third method of providing digital signatures. This method makes use of the properties of mathematical objects known as elliptic curves. Because CICS Transaction Server (TS) V3.1 does not support this method, this book does not discuss it.

2.9.4 Comparing RSA with Digital Signature Algorithm for digital signatures

In DSA, signature generation is faster than signature verification, while in the RSA algorithm, signature verification is faster than signature generation if the public and private exponents respectively, are chosen for this property, which is usually the case. It can in fact, be claimed that it is advantageous for signing to be the faster task. However, because a piece of digital information is signed once but verified often in many applications, it may well be more advantageous to have faster verification.

2.10 Public key digital certificates

The tricky part about digital signatures is the trustworthy distribution of public keys because the receiver requires a genuine copy of the sender’s public key. This is provided by public key digital certificates.

A digital certificate is analogous to a passport in the following ways:

- Passports are issued by a trusted authority such as a government passport office. Digital certificates are issued by trusted authorities known as Certificate Authorities or CAs.
- A government passport office does not issue a passport unless the persons requesting it have proven their identity and citizenship to the passport office. CAs have the responsibility of checking the credentials provided in an application for a digital certificate. The CA may, for example, require the person who is requesting the certificate to appear in person and show a birth certificate.

- A passport certifies the bearer’s name, address, and citizenship. A digital certificate establishes the subject’s distinguished name (DN) and public key.
- Specialized equipment is used in the creation of passports to make it very difficult to alter the information in it or to forge a passport. CAs sign the digital certificates they issue with their private key.
- If other authorities, such as the border police in other countries, trust the authority that issued the passport, they implicitly trust the passport. If a Web user trusts a CA, he implicitly trusts the digital certificates issued by the CA.
- Both passports and digital certificates are valid for a limited period.

The process of issuing a certificate is as follows:

1. The requester generates a public key and a private key pair and sends the public key to an appropriate CA with some proof of identification.
2. The CA checks the identification and takes any other steps that may be necessary to assure itself that the request did indeed come from the requester, and that the public key has not been modified in transit.
3. The CA then sends the requester a certificate, which attests that the public key belongs to the requester.

In the discussion on digital certificates that follows, the description of the version 3 format of a digital certificate as given in RFC 3280 Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, is used.

The certificate is a sequence of three required fields:

- tbsCertificate

  The tbsCertificate field contains the name of the certificate’s subject, a public key associated with the subject, the name of the person who issues the certificate, a validity period, and other associated information that are described in 2.10.1, “tbsCertificate” on page 54.

- signatureAlgorithm

  The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate. RFC 3279 Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL Profile lists the supported algorithms:

  - md2WithRSAEncryption

    This algorithm uses md2 for the hash function and RSA for the encryption algorithm.

  - md5WithRSAEncryption
  - sha-1WithRSAEncryption
  - id-dsa-with-sha1

This algorithm uses SHA-1 for the hash function, and uses the Digital Signature Algorithm.

– ecdsa-with-SHA1

- signatureValue

  The signatureValue field contains a digital signature computed on the tbsCertificate. The ASN.1 DER-encoded tbsCertificate is used as the input to the signature function. This signature value is encoded as a BIT STRING and included in the signature field, as shown in Figure 2-11.

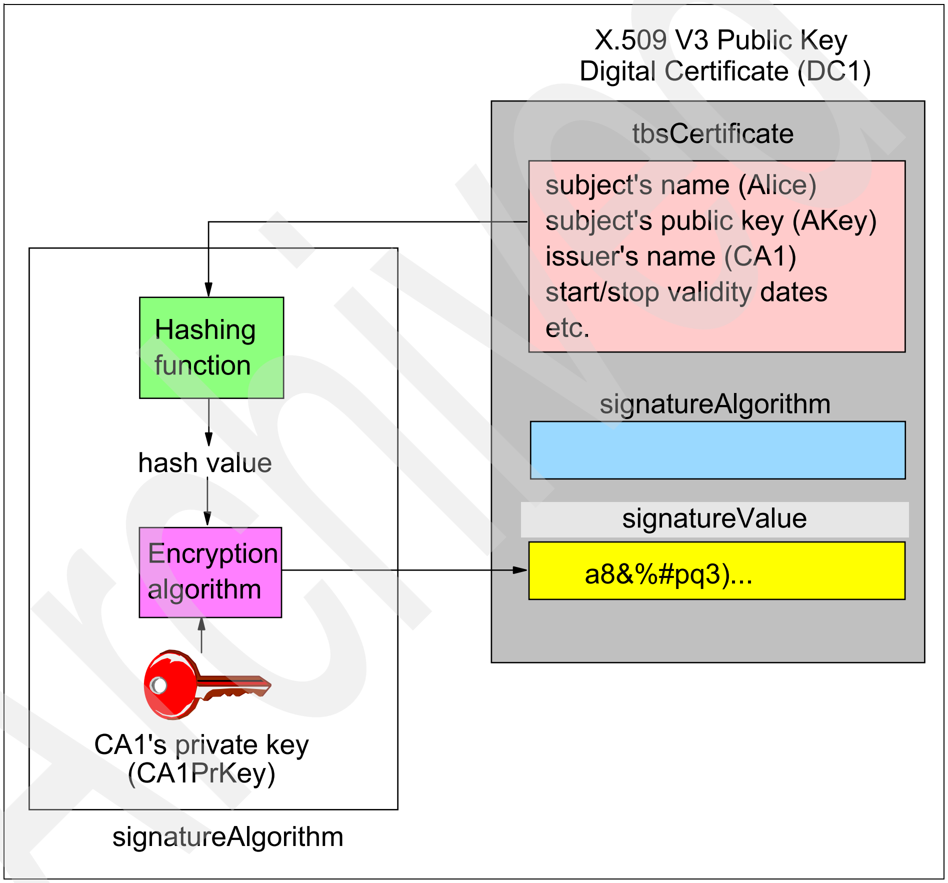


Figure 2-11 X.509 V3 public key digital certificate

By generating this signature, a CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

2.10.1 tbsCertificate

A tbsCertificate contains the following fields:

- version

  ITU-T X.509, which was first published in 1988 as part of the X.500 Directory recommendations, defines a standard certificate format. (ITU-T is the International Telecommunications Union. It was earlier known as CCITT, and is a multinational union that provides standards for telecommunications equipment and systems). The certificate format in the 1988 standard is called the V1 format. When X.500 was revised in 1993, two more fields were added, resulting in the V2 format. Experience gained in attempts to deploy RFC 1422 Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management revealed the necessity to develop a third version. In June 1996, standardization of the basic V3 format was completed. The value stored in the version field is one less than the version number, for example, when the version is 3, the value stored in the version field is 2.

- serialNumber

  The serial number is a positive integer assigned by the CA to each certificate. It is unique for each certificate issued by a CA, that is, the issuer's name and serial number identify a unique certificate.

- signature

  This field contains the algorithm identifier for the algorithm used by the CA to sign the certificate. This field must contain the same algorithm identifier as the signatureAlgorithm field.

- issuer

  The issuer field identifies the entity that has signed and issued the certificate. It contains a DN. For more details about DN, refer to "Distinguished names" on page 57.

- validity

  The certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate. The field is represented as a sequence of two dates:

  - notBefore

    This is the date on which the certificate validity period begins

  - notAfter

    This is the date on which the certificate validity period ends

  Both notBefore and notAfter may be encoded as YYMMDDHHMMSSZ or YYYYMMDDHHMMSSZ.

- subject

  The subject field identifies the entity associated with the public key stored in the subjectPublicKeyInfo field. The subject field contains a DN.

  If the subject is a CA, the subject field must contain a DN that matches the contents of the issuer field in all the certificates issued by the subject CA.

- subjectPublicKeyInfo

  This field is used to carry the public key and identify the algorithm with which the key is used. RFC 3279 lists the supported algorithm identifiers:

  - rsaEncryption

    When the algorithmIdentifier is rsaEncryption, the public key must be encoded as a sequence of two integers, the modulus n and the public exponent e.

  - id-dsa

    When the algorithmIdentifier is id-dsa, the public key must be encoded as the integer y.

  - dhpublicnumber

    This identifies the Diffie-Hellman key exchange algorithm. The public key is the integer $y = g^x \bmod p$.

  - id-keyExchangeAlgorithm

    This identifies the Key Exchange Algorithm (KEA), which is a key agreement algorithm. This book does not discuss this in detail.

  - id-ecPublicKey

    When the algorithmIdentifier is id-ecPublicKey, the public key is intended for use in either the ECDSA or the Elliptic Curve Diffie-Hellman (ECDH) key exchange algorithm. This book does not discuss either of these in detail.

- issuerUniqueId (optional)

  This field is used to handle the possibility of reuse of issuer names over time. RFC 3280 recommends that names must not be reused for different entities and that Internet certificates must not make use of unique identifiers.

- subjectUniqueId (optional)

  This field is used to handle the possibility of reuse of subject names over time. RFC 3280 recommends against the use of this field.

- extensions (optional)

  If present, this field is a sequence of one or more certificate extensions. The extensions defined for X.509 V3 certificates provide methods for associating additional attributes with users or public keys and for managing a certification hierarchy. A few of the standard extensions defined in RFC 3280 is discussed in 2.10.2, “Standard extensions for X.509 V3 digital certificates” on page 58.

Example 2-1 shows the decoding of an X.509 certificate as found in the following Web site:

http://en.wikipedia.org/wiki/X.509

Example 2-1 Sample X.509 certificate

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/Email=server-certs@thawte.com
       Validity
           Not Before: Jul 9 16:04:02 1998 GMT
           Not After : Jul 9 16:04:02 1999 GMT
       Subject: C=US, SP=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/Email=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f:
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
```

```
d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
68:9f
```

Distinguished names

Both the issuer field and the subject field of tbsCertificate must contain an X.520 DN. A DN is a sequence of Relative Distinguished Names (RDNs). An RDN™ has the form <attribute type> = <value>.

Table 2-1 shows the string representations of common attribute types.

Table 2-1 Attribute types and their string representations

Attribute type	String
countryName	C
organizationName	O
organizationalUnitName	OU
stateOrProvinceName	SP
localityName	L
commonName	CN

You can think of a DN as a unique name that unambiguously identifies a single entry in a directory information tree. Each RDN in a DN corresponds to a branch in the tree leading from the root of the tree to the directory entry. In Figure 2-12, the distinguished name C=US,O=IBM,OU=IO,SP=NY,L=End,CN=Bob Herman describes Bob Herman, who works in the village of Endicott in the state of New

York, USA, for the Integrated Operations unit of IBM. The distinguished name C=FR,O=IBM,OU=S&D,SP=Her,L=MOP,CN=Nigel Williams describes Nigel Williams, who works in the city of Montpellier in the province of Herault, France, for the Sales and Distribution unit of IBM.

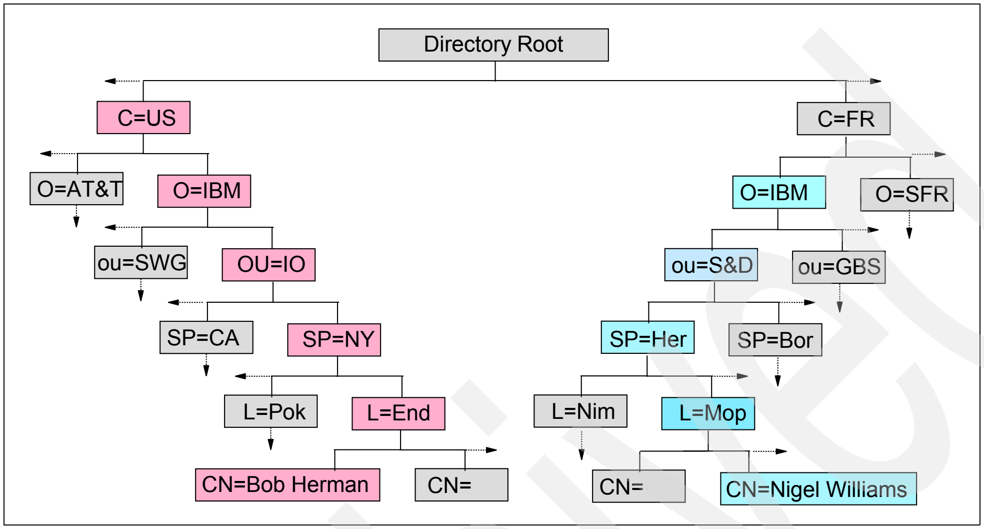


Figure 2-12 Distinguished names

2.10.2 Standard extensions for X.509 V3 digital certificates

Because RFC3280 defines 16 standard extensions, this book limits the discussion of standard extensions to those supported by the RACF RACDERT command that you can use to generate a digital certificate. These extensions are:

- Key usage

  The key usage extension defines the purpose of the subject public key contained in the certificate:

  - digitalSignature (0)

    The key is used with a digital signature mechanism to support security services other than certificate signing (bit 5) or CRL signing (bit 6). Digital signature mechanisms are often used for entity authentication and data origin authentication with integrity. (For more details about the CRL, refer to 2.11, “Certificate revocation lists” on page 63.)

  - nonRepudiation (1)

    The key is used to verify the digital signatures that are used to provide a nonrepudiation service, which protects against the signing entity falsely denying some action, excluding certificate or CRL signing. If there is a conflict later, a reliable third party may determine the authenticity of the signed data.

  - keyEncipherment (2)

    The key is used for key transport, for example, when an RSA key is to be used for key management, then this bit is set.

  - dataEncipherment (3)

    The key is used for enciphering user data other than cryptographic keys.

  - keyAgreement (4)

    The key is used for key agreement, for example, when a Diffie-Hellman key is to be used for key management, then this bit is set.

  - keyCertSign (5)

    The key is used for verifying a signature on public key certificates.

  - cRLSign (6)

    The key is used for verifying a signature on a CRL.

  - encipherOnly (7)

    The meaning of the encipherOnly bit is undefined in the absence of the keyAgreement bit. When the encipherOnly bit is asserted and the keyAgreement bit is also set, the key may be used only for enciphering data when performing a key agreement.

  - decipherOnly (8)

    The meaning of this bit is the same as the encipherOnly bit, except that it applies to a decipher operation.

  Usage restriction may be employed when a key that can be used for more than one operation is to be restricted.

- Subject alternative name

  The subject alternative name extension allows additional identities to be bound to the subject of the certificate. Following are the defined options:

  - An Internet electronic mail address
  - A domain name system (DNS) name
  - An IP address
  - A uniform resource identifier (URI)

The subject alternative name is considered to be definitively bound to the public key.

Example 2-2 shows the syntax of a RACF RACDCERT command that you can use to generate a digital certificate.

Example 2-2 RACF command for generating a digital certificate

```
RACDCERT ID(userid) GENCERT
         SUBJECTSDN(
                     CN(‘common-name’)
                     T(‘title’)
                     OU(‘organizational-unit-name1’,...)
                     O(‘organization-name’)
                     L(‘locality’)
                     SP(‘state-or-province’)
                     C(‘country’) )
         SIZE(size-of-new-private-key-in-decimal-bits)
         NOTBEFORE( DATE(yyyy-mm-dd) TIME(hh:mm:ss) )
         NOTAFTER ( DATE(yyyy-mm-dd) TIME(hh:mm:ss) )
         WITHLABEL(‘label-name’)
         SIGNWITH( CERTAUTH|SITE LABEL(‘label-name’) )
         PCICC | ICSF
         KEYUSAGE( HANDSHAKE DATAENCRYPT DOCSIGN CERTSIGN)
         ALTNAME( IP(numeric-ip-address)
                  DOMAIN(‘internet-domain-name’)
                  EMAIL(‘email-address’)
                  URI(‘universal-resource-identifier’) )
```

The values that you specify for the KEYUSAGE parameter specify the values for the KeyUsage certificate extension as follows:

- HANDSHAKE

  The key facilitates identification and key exchange during security handshakes, such as SSL. RACF sets the digitalSignature and keyEncipherment indicators in the extension.

- DATAENCRYPT

  The key is used to encrypt data. RACF sets the dataEncipherment indicator in the extension.

- DOCSIGN

  The key is used to produce a legally binding signature. RACF sets the nonRepudiation indicator in the extension.

- CERTSIGN

  The key is used to sign other digital certificates and CRLs. RACF sets the keyCertSign and cRLSign indicators in the extension.

2.10.3 Certification paths

In Figure 2-11 on page 53, the certificate authority CA1 issues a digital certificate DC1 to certify that public key AKey belongs to Alice. But how do you know that you can trust digital certificate DC1? Because DC1 is essentially the message tbsCertificate signed with CA1’s private key, the digital signature must be verified in the same way that the digital signature in Figure 2-9 on page 47 is verified. This involves using CA1’s public key to proceed as shown in Figure 2-13.

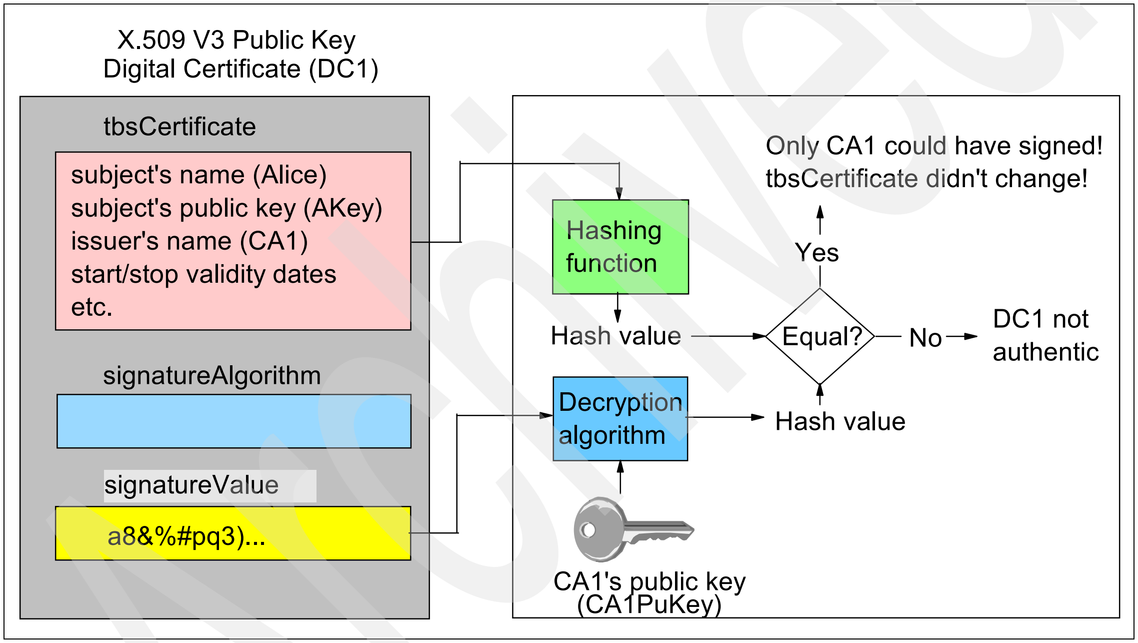


Figure 2-13 Verifying the digital certificate

However, if you do not already hold an assured copy of CA1’s public key, a digital certificate signed by another CA to certify that the CA1PuKey belongs to CA1 is required (see Figure 2-14).

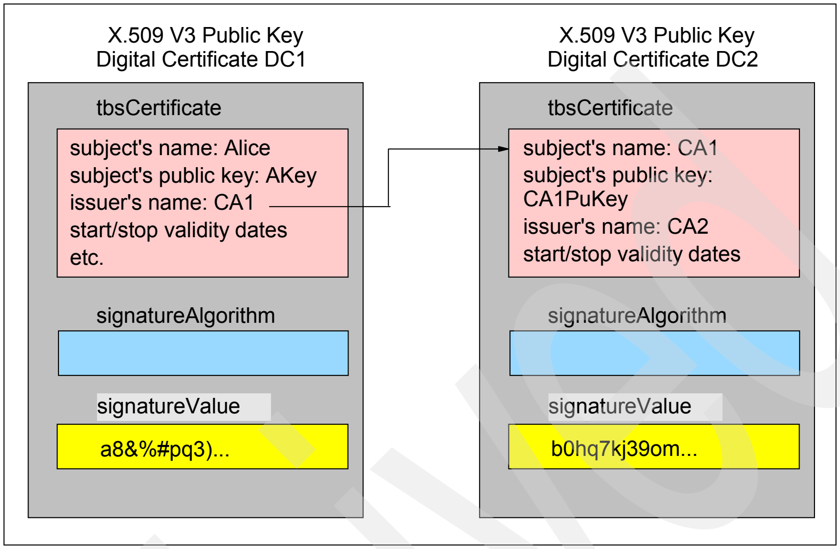


Figure 2-14 Certification path

In general, a sequence of n certificates that satisfy the following conditions are required:

- For all x in {1, 2,..., n-1}, the issuer of certificate x is the subject of certificate x+1.
- Certificate n is issued by a CA that is trusted without a certificate from any other CA. The certificate may be a self-signed certificate (one in which the CA uses its own private key to attest that the subject public key belongs to the CA).
- Certificate 1 is the certificate to be validated.
- For all x in {1,2,...n} the certificate is valid at the time in question.

Such a chain is called a certification path.

2.11 Certificate revocation lists

When a certificate is issued, it is expected to be in use for its entire validity period. However, various circumstances may cause a certificate to become invalid prior to the expiration of the validity period. Such circumstances include change of name, change of association between the subject and the CA, for example, an employee terminates employment with an organization, and compromise or suspected compromise of the corresponding private key. Under such circumstances, the CA must revoke the certificate.

X.509 defines one method of certificate revocation. This method involves each CA periodically issuing a signed data structure called a certificate revocation list (CRL). A CRL is a time-stamped list identifying the revoked certificates, which is signed by a CA, and made freely available in a public repository. Each revoked certificate is identified in a CRL by its certificate serial number. When a certificate-using system uses a certificate, that system not only checks the certificate signature and validity, but also acquires a suitably recent CRL and checks that the certificate serial number is *not* on that CRL. A new CRL is issued on a regular periodic basis. An entry is added to the CRL as part of the next update following notification of revocation.

Each CRL has a particular scope. The CRL scope is the set of certificates that could appear on a given CRL, for example, the scope could be “all certificates issued by CA X”, “all CA certificates issued by CA X”, or “all certificates issued by CA X that have been revoked for reasons of key compromise and CA compromise”.

A complete CRL lists all the unexpired certificates within its scope that have been revoked for one of the revocation reasons covered by the CRL scope. The CRL issuer may also generate delta CRLs. A delta CRL only lists those certificates within its scope whose revocation status has changed since the issuance of a referenced complete CRL (known as the base CRL).

A CRL is a sequence of three required fields:

- tbsCertList

  The tbsCertList is itself a sequence of required and optional fields:

  – version (optional)

    The version field describes the version of the encoded CRL. When the version is 2, the integer value for the field is 1.

  – signature

    The signature field contains the algorithm identifier for the algorithm used to sign the CRL.

  - issuer

    The issuer field contains an X.500 DN that identifies the entity that has signed and issued the CRL.

  - thisUpdate

    This field indicates the issue date of this CRL.

  - nextUpdate

    This field indicates the date by which the next CRL will be issued. The next CRL could be issued before the indicated date, but it will not be issued any later than the indicated date.

  - revokedCertificates (optional)

    The revoked certificate list is optional to support the case where a CA has not revoked any unexpired certificates that it has issued. The revokedCertificates field is a sequence of three fields:

    - userCertificate

      This field contains the serial number of the revoked certificate. Certificates revoked by the CA are uniquely identified by the certificate serial number.

    - revocationDate

      This field contains the date on which the revocation occurred.

    - crlEntryExtensions (optional)

      Some of the extensions for CRL entries is discussed in 2.11.1, “Extensions for entries in a certificate revocation list” on page 65.

  - crlExtensions (optional)

    Some of the extensions for CRLs is discussed in 2.11.2, “Extensions for a certificate revocation list” on page 66.

- signatureAlgorithm

  The signatureAlgorithm field contains the algorithm identifier for the algorithm used by the CRL issuer to sign the CRL.

- signatureValue

  The signatureValue field contains a digital signature computed on the tbsCertList. The ASN.1 DER encoded tbsCertList is used as the input to the signature function. This signature value is encoded as a BIT STRING and included in the CRL signatureValue field.

2.11.1 Extensions for entries in a certificate revocation list

The extensions for entries in a CRL that are defined in RFC3280 include the following:

- reasonCode

  This extension identifies the reason for the certificate revocation as follows:

  - `unspecified (0)`
  - `keyCompromise (1)`
  - `caCompromise (2)`
  - `affiliationChanged (3)`
  - `superseded (4)`
  - `cessationOfOperation (5)`
  - `certificateHold (6)`
  - `removeFromCRL (8)`
  - `privilegeWithdrawn (9)`
  - `aACompromise (10)`

  The certificateHold status is a reversible status that can be used to notice the temporary invalidity of the certificate, for example, when the user is not sure if the private key is lost. If, in this example, the private key is found again and nobody had access to it, the status can be reinstated and the certificate becomes valid again, thus removing the certificate from further CRLs.

- holdInstructionCode

  This extension indicates the action to be taken after encountering a certificate that has been placed on hold:

  - `reject`

    Reject the certificate

  - `callissuer`

    Call the certificate issuer or reject the certificate

- invalidityDate

  This extension provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry, which is the date on which the CA processed the revocation. When a revocation is first posted by a CRL issuer in a CRL, the invalidity date may precede the date of issue of the earlier CRLs.

2.11.2 Extensions for a certificate revocation list

The CRL extensions that are defined in RFC3280 include the following:

- authorityKeyIdentifier

  This extension provides a means of identifying the public key corresponding to the private key used to sign a CRL.

- issuerAltName

  This extension allows the following additional identities to be associated with the issuer of the CRL:

  - An e-mail address
  - A DNS name
  - An IP address
  - An URI

- cRLNumber

  This extension conveys a monotonically increasing sequence number for a given CRL scope and CRL issuer. It allows users to easily determine when a particular CRL supersedes another CRL. CRL numbers also support the identification of complementary complete CRLs and delta CRLs.

- issuingDistributionPoint

  This extension identifies the CRL distribution point and scope for a particular CRL and indicates whether the CRL covers revocation for end entity certificates only, CA certificates only, attribute certificates only, or a limited set of reason codes.

- freshestCRL

  This extension identifies how delta CRL information for this complete CRL is obtained.

- deltaCRLIndicator

  This extension identifies a CRL as being a delta CRL. Delta CRLs contain updates of previously distributed revocation information, rather than all the information that appears in a complete CRL. The use of delta CRLs sometimes reduces network load and processing time. The extension contains the number of the base CRL, that is, it contains the number that identifies the CRL for a given scope, which was used as the starting point in the generation of this delta CRL.

2.11.3 Security considerations when using digital certificates

RFC 3280 advises users to consider the following points when using digital certificates:

- The procedures performed by CAs to validate the binding of the subject's identity to their public key greatly affect the assurance that ought to be placed on the certificate. Different CAs may issue certificates with varying levels of identification requirements. One CA may insist on seeing a driver's license, another may want the certificate request form to be notarized, yet another may want fingerprints of anyone requesting a certificate.

  Relying parties may wish to review the CA's certificate practice statement in order to avoid situations such as the one described here. Assume that Mallory wishes to impersonate Alice. If Mallory can convincingly sign messages as Alice, he can send a message to Alice's bank stating "I wish to withdraw $10,000 from my account. Send me the money." To carry out this attack, Mallory generates a key pair and sends the public key to a CA stating "I'm Alice. Here is my public key. Please send me a certificate." If the CA believes Mallory and sends him such a certificate, Mallory can in turn fool the bank.

- The use of a single key pair for the signature and other purposes is strongly discouraged. Use of separate key pairs for the signature and key management provides several benefits to the users. The ramifications associated with the loss or disclosure of a signature key are different from that associated with the loss or disclosure of a key management key. Using separate key pairs permits a balanced and flexible response.

- The protection afforded by private keys is a critical security factor. Failure on the part of users to protect their private keys allows attackers to masquerade as them or decrypt their personal information.

- The availability and freshness of the revocation information affects the degree of assurance that ought to be placed on a certificate. If the revocation information is untimely or unavailable, the assurance associated with the binding is clearly reduced.

- The certification path validation algorithm depends on certain knowledge about the public keys and other information about one or more trusted CAs. The decision to trust a CA is an important one because it ultimately determines the trust afforded to a certificate.

- The binding between a key and a certificate subject cannot be stronger than the cryptographic module implementation and algorithms used to generate the signature. Short key lengths or weak hash algorithms limit the utility of a certificate.

2.12 The Diffie-Hellman key agreement protocol

A key agreement protocol, also called a key exchange protocol, is a protocol that allows two parties with no prior knowledge of each other to jointly establish a shared secret key over an insecure communications channel. This key can then be used to encrypt subsequent communications using a secret key algorithm.

One example of such a protocol is the Diffie-Hellman key agreement protocol, which was first published publicly by Whitfield Diffie and Martin Hellman in 1976.

The protocol has two system parameters p and g. They are both public and may be used by all the users in a system. Parameter p is a prime number, and parameter g, which is usually called a generator, is an integer less than p, with the following property:

For every number n between 1 and p-1 inclusive, there is a power k of g, such that $n = g^k$ mod p. For example, if p = 7, then g = 3 is a generator because $1 = 3^0$ mod 7, $2 = 3_2$ mod 7, $3 = 3^1$ mod 7, $4 = 3^4$ mod 7, $5 = 3^5$ mod 7, and $6 = 3^3$ mod 7.

If Alice and Bob want to agree on a shared secret key using the Diffie-Hellman key agreement protocol, they proceed as follows.

1. Alice and Bob agree upon a prime number p and a generator g.
2. Alice generates a random private integer *a* and then derives her public value g^a mod p.
3. Alice sends her public value to Bob.
4. Bob generates a random private integer *b* and then derives his public value g^b mod p.
5. Bob sends his public value to Alice.
6. Alice computes $(g^b)^a$ mod p.
7. Bob computes $(g^a)^b$ mod p.

Because $(g^b)^a$ mod p = g^{ba} mod p = g^{ab} mod p = $(g^a)^b$ mod p, Bob and Alice now have a shared secret key.

If, for example, Alice and Bob agree to use a prime number p = 23 and a generator g = 5:

1. If Alice chooses a secret integer a = 6, she computes her public value 5^6 mod 23 = 8.
2. Alice sends her public value 8 to Bob.

3. If Bob chooses a secret integer b = 15, he computes his public value 5^{15} mod 23 = 19
4. Bob sends his public value 19 to Alice.
5. Alice computes 19^{6} mod 23 = 2.
6. Bob computes 8^{15} mod 23 = 2.

Alice and Bob now have the shared secret key 2.

The Diffie-Hellman key agreement protocol as described in the previous example is vulnerable to a man-in-the middle attack. In this attack, Eve (short for eavesdropper) intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Eve substitutes it with her own and sends it to Alice. Eve and Alice thus agree on one shared key and Eve and Bob agree on another shared key. After this exchange, Eve simply decrypts any messages sent out by Alice or Bob, and then reads, and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party. This vulnerability is present because the protocol does not *authenticate* the participants. The protocol described in the previous example is sometimes called anonymous Diffie-Hellman.

The *authenticated* Diffie-Hellman key agreement protocol or Station-to-Station (STS) protocol was presented by Diffie, van Oorschot, and Wiener in 1992. Before the execution of this protocol, Alice and Bob each obtain a public or private key pair and a certificate for the public key. They also agree upon the two system parameters, p and g. The protocol then proceeds as follows:

1. Alice generates a random number a and computes and sends g^{a} to Bob.
2. Bob generates a random number b and computes g^{b}.
3. Bob computes the shared secret key $K = (g^{a})^{b}$.
4. Bob concatenates the exponentials (g^{b}, g^{a}) (the order is important), signs them using his private key B, and then encrypts them with K. He sends the ciphertext along with his own exponential g^{b} to Alice.
5. Alice computes the shared secret key $K = (g^{b})^{a}$.
6. Alice decrypts (g^{b}, g^{a}) using the shared secret key K and verifies Bob's signature using Bob's public key.
7. Alice concatenates the exponentials (g^{a}, g^{b}) (the order is important), signs them using her private key *A*, and then encrypts them with K. She sends the ciphertext to Bob.
8. Bob decrypts and verifies Alice's signature.

Alice and Bob are now mutually authenticated and have a shared secret. This secret, K, can be used to encrypt further communication.

2.13 Transport Layer Security 1.0 protocol

The primary goal of the Transport Layer Security (TLS) protocol is to provide privacy (confidentiality) and data integrity between two applications communicating over the Internet. The protocol allows client/server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

At the time of writing this book, two versions of the TLS protocol existed. The Internet Society's RFC 2246 specified the TLS 1.0 protocol in January 1999. RFC 4346 specified the TLS 1.1 protocol in April 2006. CICS TS V3.1 added support for the TLS protocol. However, because CICS TS V3.1 became generally available before TLS 1.1, CICS TS V3.1 supports only TLS 1.0. Therefore, this book limits its discussion of TLS to TLS 1.0.

TLS 1.0 is based on the SSL 3.0 Protocol Specification as published by Netscape. The differences between TLS 1.0 and SSL 3.0 are not dramatic, but they are significant enough, in that, TLS 1.0 and SSL 3.0 do not interoperate. However, TLS 1.0 does incorporate a mechanism by which a TLS implementation can back down to SSL 3.0.

2.13.1 Overview of Transport Layer Security

The TLS protocol comprises two layers, the TLS Record Protocol and the TLS Handshake Protocol. At the lowest level, layered on top of some reliable transport protocol, for example, TCP, is the TLS Record Protocol. The TLS Record Protocol provides connection security that has two basic properties:

- The connection is private.

  Secret key cryptography is used for data encryption. The keys for this secret key encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol such as the TLS Handshake Protocol.

- The connection is reliable.

  Message transport includes a message integrity check using a keyed MAC. Secure hash functions, for example, SHA-1 or MD5, are used for MAC computations.

The TLS Handshake Protocol operates on top of the TLS Record Protocol and allows the server and the client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol such as Hypertext Transfer Protocol (HTTP) transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

- The peer's identity can be authenticated using public key cryptography.
- The negotiation of a shared secret is secure. The negotiated secret is unavailable to eavesdroppers, and for any authenticated connection, the secret cannot be obtained, even by attackers who can place themselves in the middle of the connection.
- The negotiation is reliable. Attackers cannot modify the negotiation communication without being detected by the parties to the communication.

The Record Protocol takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result. The received data is decrypted, verified, decompressed, and reassembled, and then delivered to higher level clients. Figure 2-15 shows a client sending a message to a server.

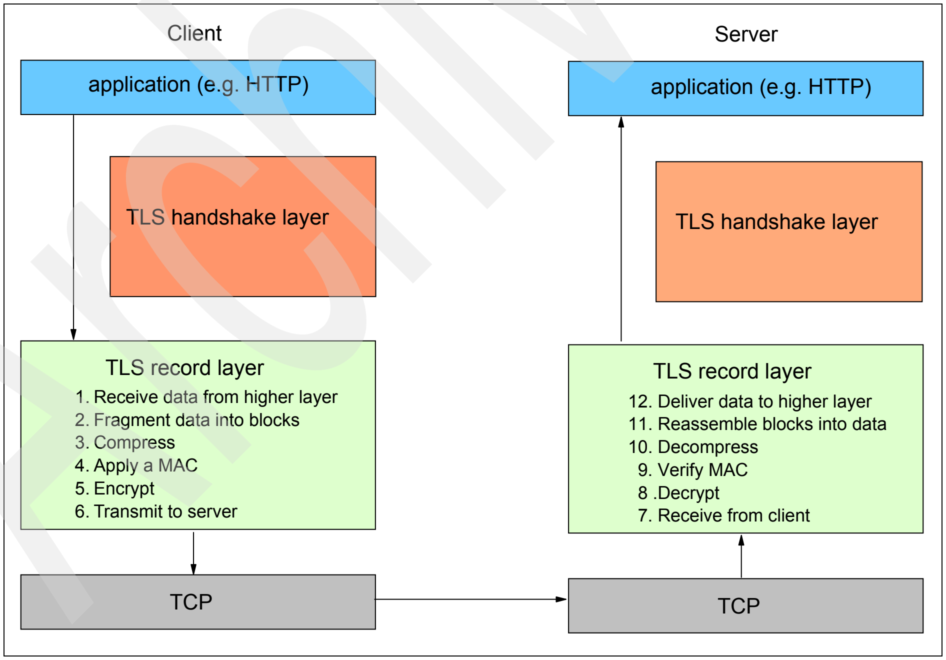


Figure 2-15 Overview of Transport Layer Security

The TLS Handshake Protocol consists of a suite of three subprotocols:

- Change cipher spec protocol
- Alert protocol
- Handshake protocol

Before describing the Handshake protocol, this chapter describes a cipher suite.

2.13.2 Cipher suites

One dictionary defines a *suite* as “a group of things forming a unit or constituting a collection”. A *cipher suite* then is a collection of cipher algorithms. More specifically, RFC 2246 defines a cipher suite as a collection consisting of one key exchange algorithm, one encryption algorithm, and one hash algorithm.

The hash algorithm must come from the following set:

- NULL (no hash algorithm)
- MD5
- SHA (meaning SHA-1)

The encryption algorithm must come from the following set:

- NULL (no encryption)
- IDEA_CBC

  IDEA is a 64-bit block cipher designed by Xuejia Lai and James Massey. It uses a 128-bit key. IDEA_CBC is IDEA running in a cipher block chaining mode.

- RC2_CBC_40

  RC2 is a variable key-size block cipher designed by Ronald Rivest for RSA Security. It uses a 64-bit block size. RC2_CBC_40 is RC2 running with a 40-bit key in cipher block chaining mode. (“RC” stands for “Ron’s Code” or “Rivest’s Cipher”.)

- RC4_40

  RC4 is a variable key-size stream cipher designed by Rivest for RSA Security. RC4_40 is RC4 running with a 40-bit key.

- RC4_128

  RC4_128 is RC4 running with a 128-bit key. When RFC 2246 was published, RC4_40 was “exportable”, but RC4_128 was not. (For many years, the US government did not approve the export of cryptographic products unless the key size was strictly limited.)

- DES40_CBC

  DES40_CBC is DES running with a 40-bit key in cipher block chaining mode.

- DES_CBC

  DES_CBC is DES running with a 56-bit key in cipher block chaining mode. When RFC 2246 was published, DES40_CBC was exportable, but DES_CBC was not.

- 3DES_EDE_CBC

  3DES_EDE_CBC is TDEA running in cipher block chaining mode. The first use of DES is for encryption, the second for decryption, and the third for encryption.

The key exchange algorithm must come from the following set:

- DHE_DSS
- DHE_DSS_EXPORT
- DHE_RSA
- DHE_RSA_EXPORT
- DH_anon
- DH_anon_EXPORT
- DH_DSS
- DH_DSS_EXPORT
- DH_RSA
- DH_RSA_EXPORT
- NULL
- RSA
- RSA_EXPORT

DH denotes key exchange algorithms in which the server's certificate contains the Diffie-Hellman parameters signed by a CA. DHE denotes ephemeral Diffie-Hellman, where the Diffie-Hellman parameters are signed by a DSS or RSA certificate, which in turn has been signed by a CA. The signing algorithm used is specified after the DH or the DHE parameter.

DH_anon indicates completely anonymous Diffie-Hellman communications in which neither party is authenticated. RSA indicates that the server must provide a RSA certificate that can be used for key exchange. (RSA certificate is an X.509 certificate that has been signed by using the RSA algorithm.)

RFC 2246 assigns names to the 27 cipher suites that contain an acceptable combination of algorithms from the sets listed earlier. The names have the following form:

```
TLS_key-exchange-algorithm_WITH_encryption-algorithm_hash-algorithm
```

For example, the cipher suite TLS_RSA_WITH_DES_CBC_SHA contains the RSA key exchange algorithm, the DES_CBC encryption algorithm, and the SHA hash algorithm. The cipher suite TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA contains the DH_DSS key exchange algorithm, the 3DES_EDE_CBC encryption algorithm, and the SHA hash algorithm.

RFC 2246 also assigns a number to each of the 27 cipher suites. Table 2-2 shows the first ten.

Table 2-2 Some TLS 1.0 cipher suite numbers and their meanings

Number	Name
01	TLS_RSA_WITH_NULL_MD5
02	TLS_RSA_WITH_NULL_SHA
03	TLS_RSA_EXPORT_WITH_RC4_40_MD5
04	TLS_RSA_WITH_RC4_128_MD5
05	TLS_RSA_WITH_RC4_128_SHA
06	TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5
07	TLS_RSA_WITH_IDEA_CBC_SHA
08	TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
09	TLS_RSA_WITH_DES_CBC_SHA
0A	TLS_RSA_WITH_3DES_EDE_CBC_SHA

The TLS 1.0 protocol seeks to provide a framework into which the new public key and secret key encryption methods can be incorporated. This prevents the necessity to create a new protocol, which may introduce weaknesses. The TLS 1.0 protocol allows additional cipher suites to be registered by publishing an RFC that specifies the cipher suite. Indeed, several such RFCs have been published, including the following:

- RFC 2712: Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)
- RFC 3268: Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)
- RFC 4132: Addition of Camellia Cipher Suites to Transport Layer Security (TLS)

- RFC 4162: Addition of SEED Cipher Suites to Transport Layer Security (TLS)
- RFC 4279: Pre-shared Key Ciphersuites for Transport Layer Security (TLS)
- RFC 4492: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)

CICS TS V3.1 adds support for the cipher suites (shown in Table 2-3) added by RFC 3268.

Table 2-3 Two cipher suites added by RFC 3268 and supported by CICS

Number	Name
2F	TLS_RSA_WITH_AES_128_CBC_SHA
35	TLS_RSA_WITH_AES_128_CBC_SHA

2.13.3 Alert protocol

Error handling in the TLS Handshake protocol is simple. When an error is detected, the detecting party sends a message called an alert message to the other party. An alert message conveys the severity of the message and a description of the alert.

The severity of the message must be one of the following:

- `warning` (1)
- `fatal` (2)

On transmission or receipt of a `fatal` alert message, both parties immediately close the connection. The servers and the clients are required to forget any session identifiers, keys, and secrets associated with a failed connection.

The description of the alert must be one of the following:

- `close_notify` (0)
- `unexpected_message` (10)
- `bad_record_mac` (20)
- `decryption_failed` (21)
- `record_overflow` (22)
- `decompression_failure` (30)
- `handshake_failure` (40)
- `bad_certificate` (42)
- `unsupported_certificate` (43)
- `certificate_revoked` (44)
- `certificate_expired` (45)
- `certificate_unknown` (46)
- `illegal_parameter` (47)

- unknown_ca (48)
- access_denied (49)
- decode_error (50)
- decrypt_error (51)
- export_restriction (60)
- protocol_version (70)
- insufficient_security (71)
- internal_error (80)
- user_canceled (90)
- no_renegotiation (100)

RFC 2246 provides a short explanation of each of these descriptions.

The user-canceled and no-renegotiation alerts carry a level of warning. Senders may determine at their discretion whether the following alerts are fatal or not:

- bad_certificate
- unsupported_certificate
- certificate_revoked
- certificate_expired
- certificate_unknown
- decrypt_error

The remaining alerts always carry a level of fatal.

2.13.4 Handshake protocol

When a TLS client and server first start communicating, they agree on which version of the TLS protocol they will use, select a cipher suite, optionally authenticate each other, and use public key encryption techniques to generate shared secrets.

The TLS Handshake Protocol involves the following steps:

1. Exchange hello messages to agree on a cipher suite and a compression algorithm, exchange random values, and check for session resumption
2. Exchange the necessary cryptographic parameters to allow the client and server to agree on a premaster secret
3. Exchange certificates and cryptographic information to allow the client and the server to authenticate themselves
4. Generate a master secret from the premaster secret and exchanged random values
5. Provide security parameters to the record layer, as shown in Figure 2-16

6. Allow the client and server to verify whether their peer has calculated the same security parameters and whether the handshake occurred without tampering by an attacker

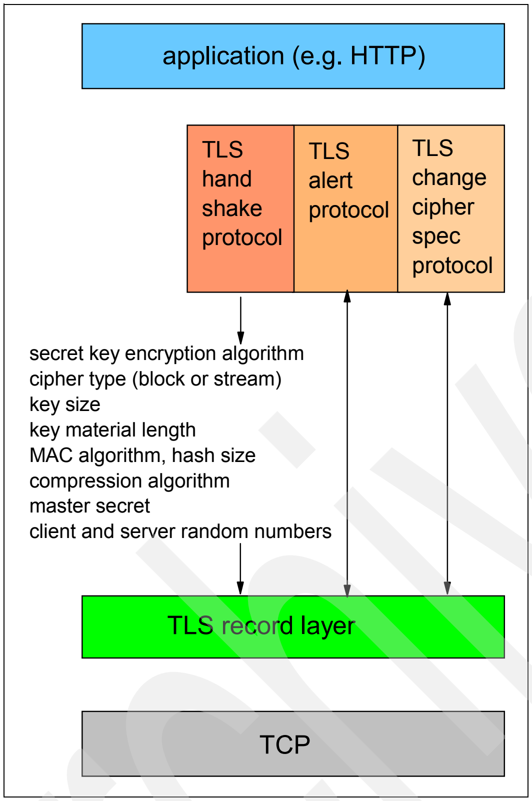


Figure 2-16 Handshake protocol passes security parameters to the record layer

Starting a new session

When the client and server want to start a new session, they begin by exchanging hello messages, as shown in Figure 2-17.

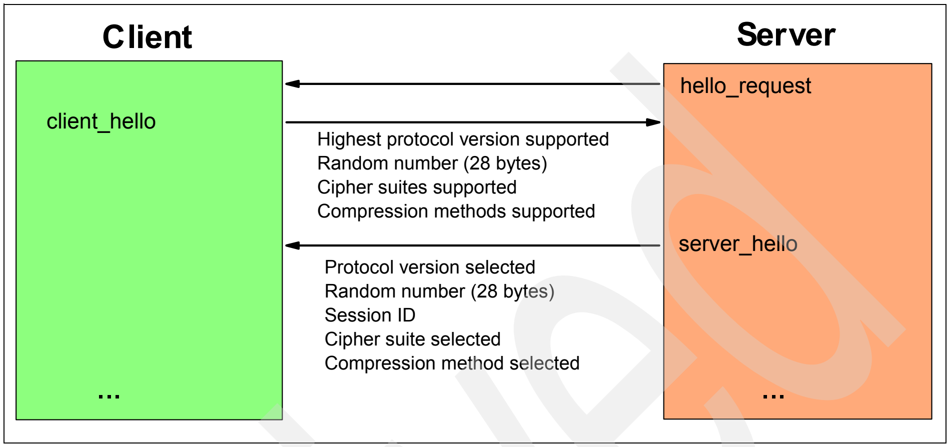


Figure 2-17 Starting a new TLS session(1): Establishing algorithms

The server may send the `hello_request` message at any time. It is a simple notification and the client must begin the negotiation process anew by sending a `client_hello` message when convenient.

The `client_hello` message includes the following parameters:

- The version of the TLS protocol by which the client wishes to communicate during this session. This must be the highest valued version supported by the client. For TLS 1.0, the version must be 3.1.
- The current time and date according to the client's internal clock, followed by 28 bytes generated by a secure random number generator.
- A list of the cipher suites supported by the client in the order of the client's preference (with the favorite choice coming first). Each cipher suite defines a key exchange algorithm, a secret key encryption algorithm, including the secret key length, and a MAC algorithm.
- A list of the compression methods supported by the client and sorted by the client preference.

The server sends a `server_hello` message in response to a `client_hello` message after it finds an acceptable set of algorithms. If it cannot find such a match, it responds with a `handshake_failure` alert. The `server_hello` message includes the following parameters:

- The lower of the TLS protocol version suggested by the client and the highest TLS protocol version supported by the server
- The current time and date according to the server's internal clock, followed by 28 bytes generated by a secure random number generator
- The identity of the session corresponding to this connection. The actual contents of the sessionID are defined by the server.
- The single cipher suite selected by the server from the list supplied by the client
- The single compression algorithm selected by the server from the list supplied by the client

Thus the `client_hello` and `server_hello` messages establish the following connection attributes:

- The TLS protocol version
- The sessionID
- The key exchange algorithm
- The secret key encryption algorithm
- The key length for the secret key encryption algorithm
- The compression method

Additionally, two random values are generated and exchanged:

`client_hello.random` and `server_hello.random`.

When this list of items is compared with the list of items shown in Figure 2-16 on page 77, which the TLS Handshake Protocol must pass to the Record layer, you can see that the TLS Handshake Protocol must still come up with a master secret.

The master secret is generated by using, among other things, a premaster secret. The general goal of the key exchange process shown in Figure 2-18 is to create a premaster secret known to the communicating parties and not to the attackers. Note that the italicized blue lines in Figure 2-18 represent the actions taken by the client and the server rather than the protocol messages.

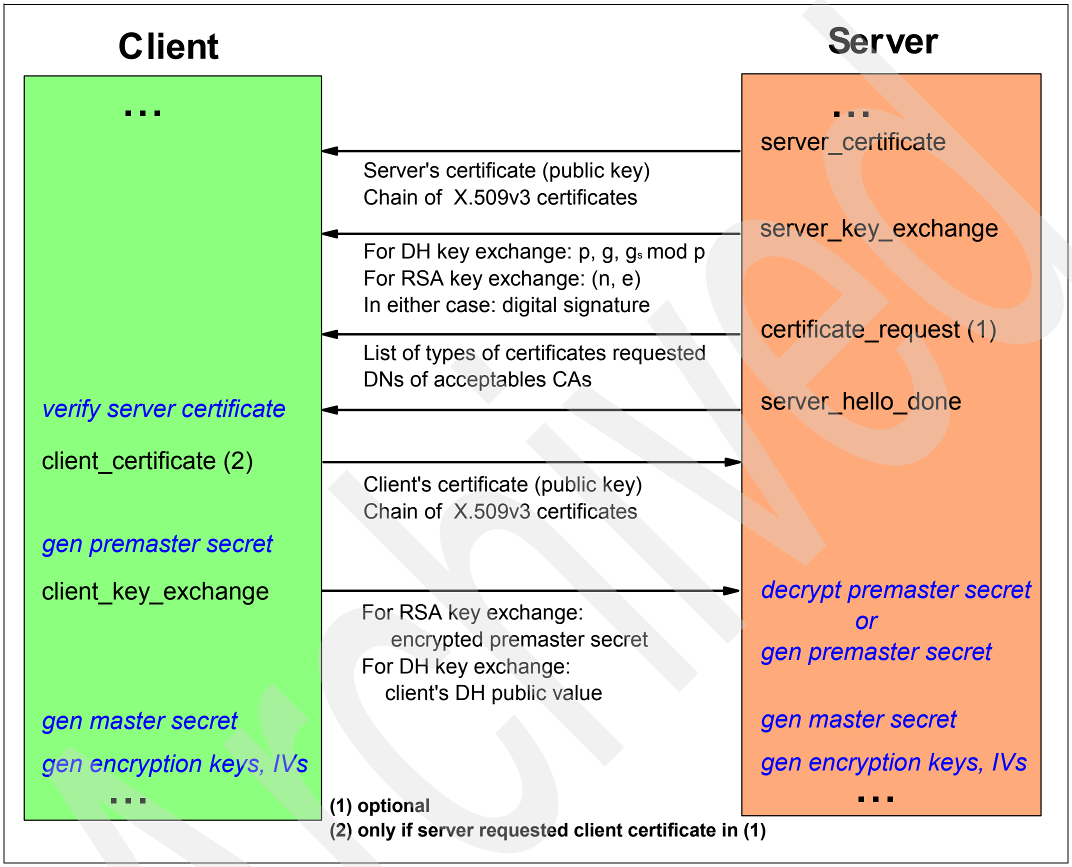


Figure 2-18 Starting a new TLS session(2): Establishing the premaster secret

The `server_certificate` message sends a chain of X.509v3 certificates. The server's certificate must come first in the chain. Each following certificate must directly certify the one preceding it. The server's certificate must contain a key that matches the key exchange method, as shown in Table 2-4.

Table 2-4 Key exchange methods and certificate key types

Key exchange method	**Certificate key type**
RSA	RSA public key. The certificate must allow the key to be used for encryption.
RSA_EXPORT	RSA public key of length greater than 512 bits that can be used for signing, or a key of 512 bits or shorter that can be used for either encryption or signing
DHE_DSS	DSS public key
DHE_DSS_EXPORT	DSS public key
DHE_RSA	RSA public key that can be used for signing
RHE_RSA_EXPORT	RSA public key that can be used for signing
DH_DSS	Diffie_Hellman key. The algorithm used to sign the certificate must be DSS.
DH_RSA	Diffie_Hellman key. The algorithm used to sign the certificate must be RSA.

Note: At the time RFC 2246 was written, United States export restrictions limited the RSA keys used for encryption to 512 bits, but did not place any limit on lengths of RSA keys used for signing operations.

The `server_key_exchange` message is sent by the server only when the `server_certificate` message does not contain enough data to allow the client to exchange a premaster secret. This is true for the following key exchange methods:

- RSA_EXPORT (if the public key in the server certificate is longer than 512 bits)
- DHE_DSS
- DHE_DSS_EXPORT

- DHE_RSA
- DHE_RSA_EXPORT
- DH_anon

The `server_key_exchange` message conveys cryptographic information to allow the client to communicate the premaster secret, either an RSA public key with which to encrypt the premaster secret or a Diffie-Hellman public key with which the client can complete a key exchange (the result being the premaster secret).

When the key exchange method is RSA_EXPORT, the `server_key_exchange` message includes the following parameters:

- The modulus n of the server's temporary RSA key

  According to US export law, at the time RFC 2246 was written, RSA moduli larger than 512 bits could not be used for key exchange in the software exported from the US. This message allows the larger RSA keys encoded in certificates to be used to sign the temporary, shorter RSA keys.

- The public exponent e of the server's temporary RSA key
- A 36-byte structure of two hashes (one SHA-1 and one MD5) that has been signed with the server's private key. The SHA-1 hash takes as input the concatenation of client_hello.random, server_hello.random, and (n,e). It produces 20 bytes of output. The MD5 hash takes the same input and produces 16 bytes of output.

When the key exchange method is DHE_DSS, DHE_DSS_EXPORT, DHE_RSA, or DHE_RSA_EXPORT, the `server_key_exchange` message includes the following parameters:

- The prime modulus p used for the Diffie-Hellman operation
- The generator g used for the Diffie-Hellman operation
- The server's Diffie-Hellman public value gs mod p
- Two integers r and s produced as follows:

  An SHA-1 hash takes as input the concatenation of client_hello.random, server_hello.random, p, g, and ga mod p. It produces 20 bytes of output. The 20 bytes are run through the DSA.

A nonanonymous server can optionally request a certificate from the client. The `certificate_request` message includes the following parameters:

- A list of the types of certificates requested, sorted in the order of the server's preference
- A list of DNs of acceptable CAs

The server sends the `server_hello_done` message to indicate that it is done with sending messages to support the key exchange, and the client can proceed with its phase of the key exchange. On receipt of this message, the client must verify that the server has provided a valid certificate and that the certificate has not expired or been revoked. The client must also check that the server hello parameters are acceptable.

The `client_certificate` message is the first message that the client can send after receiving the `server_hello_done` message. The client only sends the `client_certificate` message if the server requests a certificate. If the client does not have a suitable certificate to send to the server, it sends a message containing no certificates. If the server requires client authentication in order to continue the handshake, it may respond with a `fatal_handshake` failure alert.

The structure of the `client_key_exchange` message depends on which key exchange method is selected:

- If RSA is being used for key agreement and authentication, the client generates a 48-byte premaster secret, encrypts it using either the public key from the server's certificate or the temporary RSA key provided in a `server_key_exchange` message, and then sends the result in an encrypted premaster secret message. Because the premaster secret has been encrypted using the server's public key, the server can decrypt it. In fact, only the server can decrypt it. The premaster secret consists of two bytes that indicate the latest (newest) version of the TLS protocol supported by the client, followed by 46 securely generated random bytes.
- If Diffie-Hellman is being used for key agreement, the `client_key_exchange` message conveys the client's Diffie-Hellman public value g^c mod p. Having the client's Diffie-Hellman public value allows the server to compute the same premaster secret as the client. (In the event that the key exchange method is DH_RSA or DH_DSS, and the server requests client certification and the client is able to respond with a certificate containing a Diffie-Hellman public key whose group and generator matched those specified by the server in its certificate, then the client sends an empty `client_key_exchange` message.)

Now that the client and the server have agreed on the premaster secret, they can compute the master secret. For all key exchange methods, the same algorithm is used to convert the premaster secret into the master secret.

In Example 2-3, PRF is a pseudorandom function defined in RFC 2246, and + represents the concatenation operation. PRF takes a secret such as our premaster secret as input, an identifying label such as “master secret”, and a seed such as the concatenation of the random numbers generated by the client and the server.

Example 2-3 Computing the master secret

```
master_secret=PRF(pre_master_secret, “master secret”,
                  client_hello.random+server_hello.random)
```

Having computed the master secret, the Record Protocol layer for the client and the Record Protocol layer for the server can now each use PRF to compute a key_block, as shown in Example 2-4.

Example 2-4 Computing the key_block

```
key_block=PRF(master_secret, “key expansion”,
                  client_hello.random+server_hello.random)
```

The key_block is then partitioned as follows:

- client_write_MAC_secret

  The first SecurityParameters.hash_size bytes of the key_block become the secret data used to authenticate the data written by the client.

- server_write_MAC_secret

  The next SecurityParameters.hash_size bytes of the key_block become the secret data used to authenticate the data written by the server.

- client_write_key

  The next SecurityParameters.key_material_length bytes become the key used to encrypt the data written by the client.

- server_write_key

  The next SecurityParameters.key_material_length bytes become the key used to encrypt the data written by the server.

- client_write_IV

  The next bytes become the initialization vector for the encryption algorithm when the client encrypts the data. The required number of bytes is equal to the block size for block ciphers and zero for stream ciphers.

- server_write_IV

  The next bytes become the initialization vector for the encryption algorithm when the server encrypts the data.

> **Note:** Because the client_hello.random and server_hello.random values are unique for each connection, the data encryption keys and MAC secrets are unique for each connection. Also note that the server_write_key and the client_write_key are independent of each other.

Figure 2-19 shows the final phase of starting a new TLS session.

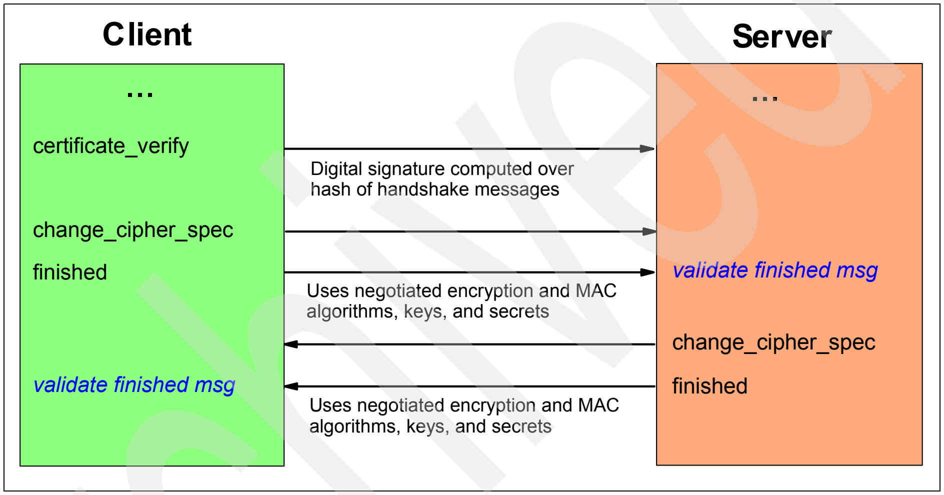

Client
Server
...
...
certificate_verify
Digital signature computed over hash of handshake messages
change_cipher_spec
finished
Uses negotiated encryption and MAC algorithms, keys, and secrets
validate finished msg
change_cipher_spec
finished
validate finished msg
Uses negotiated encryption and MAC algorithms, keys, and secrets

Figure 2-19 Starting a new TLS session (3): Verification

The `certificate_verify` message is used to provide explicit verification of a client certificate. This message follows only a client certificate that has signing capability, that is, all the certificates except those containing fixed Diffie-Hellman parameters.

When the key exchange method is RSA, the `certificate_verify` message includes a 36-byte structure of two hashes (one SHA-1 and one MD5), which has been signed with the client's private key. The SHA-1 hash takes as input the concatenation of all handshake messages sent or received starting at `client_hello` up to, but not including this message. It produces 20 bytes of output. The MD5 hash takes the same input and produces 16 bytes of output. These handshake messages include the server certificate that binds the signature to the server, and `server_hello.random` that binds the signature to the current handshake process.

When the key exchange method is Diffie-Hellman, the `certificate_verify` message includes a SHA-1 hash that takes the input described in the preceding paragraph and produces 20 bytes of output. The 20 bytes are then signed using the DSA.

The `change_cipher_spec` message is sent by both the client and the server to notify the receiving party that subsequent records will be protected under the newly negotiated encryption and MAC algorithms and keys. The message consists of a single byte of value 1.

A `finished` message is always sent immediately after a `change_cipher_spec` message to verify that the key exchange and authentication processes are successful. The `finished` message is protected first with the just-negotiated algorithms, keys, and secrets. The contents of the `finished` message is generated using PRF, as shown in Example 2-5.

Example 2-5 Computing verify_data

```
PRF(master_secret, finished_label,
    MD5(handshake_messages)+SHA-1(handshake_messages))
```

In Example 2-5:

- The value of `finished_label` is the string “client finished” for the `finished` messages sent by the client and ”server finished” for the `finished` messages sent by the server.
- The value of `handshake_messages` is all the data from all the handshake messages up to, but not including this message.

Recipients of the `finished` messages must verify that the contents are correct. After one side sends its `finished` message and then receives and validates the `finished` message from its peer, it can begin to send and receive the application data over the connection.

Outgoing data is protected with a MAC before transmission. To prevent message replay or modification attacks, the MAC is computed from the MAC secret, the message contents, the message length, and the sequence number of the message.

Resuming a session

Cryptographic operations tend to be highly CPU-intensive, particularly public key operations. For this reason, the TLS protocol has incorporated an optional session caching scheme to reduce the number of connections that have to be established from scratch. When the client and the server decide to resume a previous session, the message flow is as shown in Figure 2-20.

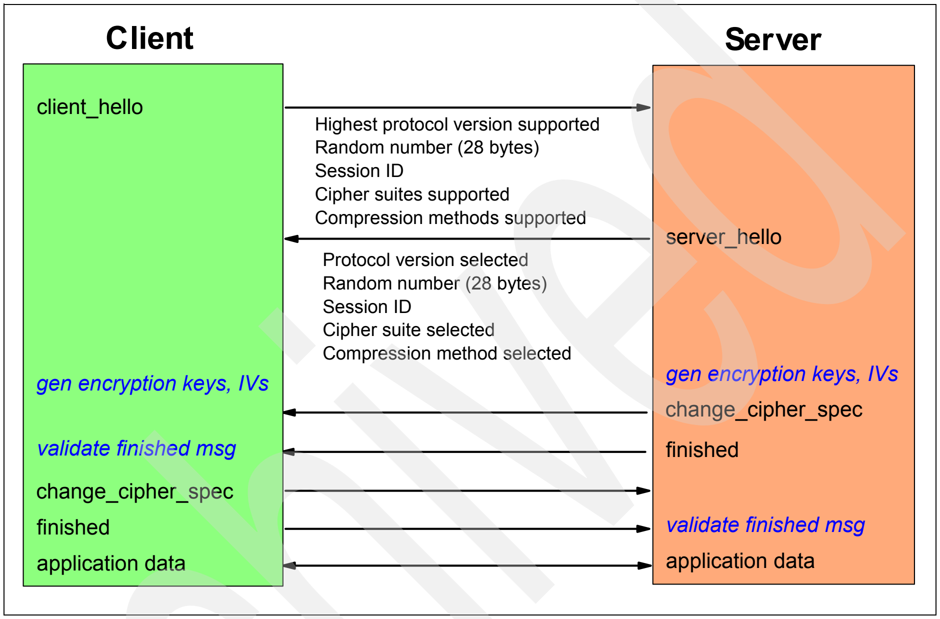


Figure 2-20 Resuming a session

The client sends a `client_hello` message using the session ID of the session to be resumed. The server then checks its session cache for a match.

- If a match is found and the server is willing to re-establish the connection under the specified session state, it sends a `server_hello` message with the same session ID value plus the cipher suite and compression method from the state of the session being resumed. At this point, both the client and the server must send `change_cipher_spec` messages and proceed directly to the `finished` messages.
- If a session ID match is not found, the server generates a new session ID and the TLS client and the server perform a full handshake.

When a connection is established by resuming a session, new client_hello.random and server_hello.random values are used with the session's master secret to produce a new key_block (Example 2-4). Thus, the new encryption keys and MAC secrets.

2.14 Cryptographic hardware

A cryptographic hardware feature is a secure, high-speed device that performs cryptographic functions. The cryptographic hardware features available to your CICS regions depend on the server you have.

This section provides a summary of the cryptographic hardware features currently available on the latest System z hardware.

2.14.1 CP Assist for Cryptographic Functions

CP Assist for Cryptographic Functions (CPACF) is a set of cryptographic instructions available on all CPs of an IBM System z9 109, IBM System z9 Enterprise Class (EC), and IBM System z9 Business Class (BC), z990, z890. Using CPACF instructions improves performance.

CPACF offers a set of symmetric cryptographic functions. The CPACF feature provides hardware acceleration for DES, triple-DES, AES (128-bit), MAC, SHA-1, and SHA-256 cryptographic services. It provides high-performance hardware encryption, decryption, and hashing support.

The SHA-1 algorithm is always available. The SHA-256 algorithm is available on the System z9 109, System z9 EC, and System z9 BC. CPACF DES/triple-DES enablement is provided with feature 3863. It provides for clear key DES and triple DES instructions. On the z9-109, z9 EC, and z9 BC, this feature includes clear key AES for 128-bit keys.

Note: The CPACF operates with *clear keys* only. A clear key is a key that has *not* been encrypted under another key and has no additional protection within the cryptographic environment.

The CPACF feature supports z/OS applications and subsystems such as CICS that use the Integrated Cryptographic Service Facility (ICSF) for cryptographic functions (see 2.15, “Integrated Cryptographic Service Facility” on page 90).

2.14.2 Crypto Express2 Feature

An installation can configure the Crypto Express2 Feature (CEX2) as an asynchronous cryptographic coprocessor or accelerator. It is only available on the System z9 109, System z9 EC, and System z9 BC.

- The Crypto Express 2 Coprocessor (CEX2C) feature on System z9 enables the user to perform the following by using secure keys:

  > **Note:** A *secure key* is a key that has been encrypted under another key, usually the master key.

  - Encrypt and decrypt data utilizing shared secret key algorithms
  - Generate, install, and distribute cryptographic keys securely using both the public and secret key cryptographic methods
  - Generate, verify, and translate personal identification numbers (PINs)
  - Ensure the integrity of data by using MACs, hashing algorithms, and RSA public key algorithm (PKA) digital signatures

  The CEX2C consolidates the functions previously offered on the z900 by the Cryptographic Coprocessor feature (CCF), the PCI Cryptographic Coprocessor (PCICC), and the PCI Cryptographic Accelerator (PCICA).

- The Crypto Express2 Accelerator (CEX2A) is actually a CEX2C that has been reconfigured by the user to only provide a subset of the CEX2C functions at enhanced speed.

  The CEX2A is used for the following RSA cryptographic operations (with clear keys only):

  - PKA Decrypt (CSNDPKD), with PKCS-1.2 formatting
  - PKA Encrypt (CSNDPKE), with ZERO-PAD formatting
  - Digital Signature Verify

For detailed information about configuring these cryptographic hardware features, refer to *z9-109 Crypto and TKE V5 Update*, SG24-7123.

> **Important:** The CEX2 feature requires ICSF to be active.

2.15 Integrated Cryptographic Service Facility

The Integrated Cryptographic Service Facility (ICSF) is a software element of z/OS that works with cryptographic hardware features and RACF to provide secure, high-speed cryptographic services in the z/OS environment. ICSF provides the application programming interfaces (APIs) by which applications and subsystems such as CICS request the cryptographic services.

ICSF provides support for a number of cryptography services, including the following:

- DES and triple DES encryption for privacy
- The transport of symmetric data keys through the use of the RSA public key algorithm
- The generation and verification of digital signatures through the use of both the RSA and the DSA algorithms
- The generation of RSA and DSA keys
- The PKA Encrypt and PKA Decrypt callable services that can be used to enhance the security and performance of Secure Sockets Layer/Transport Layer Security (SSL/TLS) security protocol applications
- Advanced Encryption Standard (AES) encryption and decryption

The CICS support for Web Services Security (WS-Security) is dependent on these ICSF services, and therefore, the configuration and startup of ICSF is a requirement for using this support.

Important: ICSF must be configured with cryptographic devices and started in order to use the CICS WS-Security support.

For general information about ICSF, refer to *z/OS V1R8.0 Cryptographic Services ICSF Overview*, SA22-7519.

2.15.1 Cryptographic hardware requirements for CICS WS-Security

Because CICS relies on ICSF for cryptographic services and ICSF is dependent on cryptographic devices to perform the actual cryptographic functions, there is a clear dependency on the availability of cryptographic devices when using the CICS WS-Security support. Specific requirements depend on the server that you have.

Table 2-5 contains a list of the ICSF callable services used by the CICS WS-Security support, and a brief summary of each service.

Table 2-5 ICSF callable services used by CICS WS-Security support

ICSF Service	Description
CSNBCKM	Multiple clear key import callable service in order to import a clear single-length, double-length, or triple-length DATA key that is to be used to encipher or decipher data
CSNBDEC	Decipher callable service in order to decipher data in an address space or a data space using the cipher block chaining mode
CSNBENC	Encipher callable service to encipher data in an address space or a data space using the cipher block chaining mode
CSNBOWH	One-way hash generate callable service in order to generate a one-way hash on specified text
CSNBRNG	Service to generate a random number
CSNBSYD	Symmetric key decipher callable service in order to decipher data in an address space or a data space using the cipher block chaining or electronic code book modes
CSNBSYE	Symmetric key encipher callable service in order to encipher data in an address space or a data space using the cipher block chaining or electronic code book modes
CSNDDSG	Digital signature generate callable service in order to generate a digital signature using a PKA private key
CSNDDSV	Digital signature verify callable service in order to verify a digital signature using a PKA public key
CSNDPKB	Service to build external PKA key tokens containing unenciphered private RSA or DSS keys
CSNDPKD	Service to decrypt (unwrap) a formatted key value. The service unwraps the key, deformats it, and returns the deformatted value to the application in the clear.

Many of these services require cryptographic hardware in order to be configured.

Important: On a z9, an optimal cryptographic hardware configuration is a combination of CPACF and CEX2.

The publication *z/OS V1R8.0 Cryptographic Services ICSF Application Programmer's Guide*, SA22-7522 documents the cryptographic hardware requirements for the System z9 and other types of servers.

3

Security technologies

This chapter examines the security technologies as they relate to the CICS Transaction Server V3.1 (z/OS), including discussions on security risks, Resource Access Control Facility (RACF), and the z/OS Security Server.

3.1 Security risks

The Internet is revolutionizing the way companies communicate and conduct business. By nature, it is public, distributed, connected, and dynamic, with the resulting effect of phenomenal growth in infrastructure, number of people online, and number and types of applications running across it. This growth, specifically the rush that governs the introduction of most new projects, gives rise to a new class of risks:

- Today, security risks are often perceived as coming from malicious hackers. These are people who are typically motivated by the mere challenge of highlighting companies' security weaknesses and winning the respect of their fellow hackers, sometimes called “bragging rights”. They are usually highly skilled, and openly communicate breakthroughs to each other, fostering the rapid development and spreading of attack tools.
- A second risk comes from the fact that a high percentage of the individuals involved in building and maintaining Web sites and the systems that support these servers are not well trained in security. The infrastructure of the Internet is complicated, and today's security solutions are usually effective only when they are tailored to the unique elements of an installation.
- A third risk, and perhaps the most serious, comes from the employees (or former employees) of a company. The Internet gives these employees the ability to work remotely and anonymously. To protect against such attacks is not easy, but a good policy is that no one person must be allowed to code, install, and use software that accesses real business data. This reduces the risk by requiring collaboration between two or three people to obtain the necessary knowledge in order to access data fraudulently.

3.1.1 Types of attacks

In the context of the CICS Web environments discussed in this book, the following five possible types of attack have been identified.

Denial-of-service attacks

The aim of a denial-of-service attack is to disable a Web site by bombarding it with requests at a rate that consumes all the available computing resources, thus denying the services to other users. Because the network connection is usually the limiting factor in Internet communication, these attacks are often mounted by planting rogue applications in the computers of educational or other institutions with high bandwidth connections, but poor security. At the time of writing this book, a spree of such distributed denial-of-service attacks on high-profile US Web sites has focused public attention on this issue. Unfortunately, this is not the only attack our systems must guard against.

Running applications without authorization

The ease-of-use with which the Internet can be accessed means that it is all too easy for someone to “crack” your system by trying to run applications without authorization. Although knowledge about cracking CICS Web applications may not be as widespread as that for cracking UNIX® systems, you must guard against the possibility. Most such attacks on UNIX systems involve misusing the ability of Web servers to run Common Gateway Interface (CGI) programs. This, of course, is not possible in a CICS system. However, there are powerful CICS transactions (such as CECI or CEMT) that can cause as much damage.

Misusing the existing applications

Most CICS applications have been developed over a period of many years and may contain myriad methods of being invoked. A disgruntled and experienced former employee could use such knowledge to invoke a function in an application by calling the CICS application with the correct input parameters.

Masquerading as an authenticated user

After you open your CICS system for use through the Internet, you can no longer rely on the physical security of your terminals (the way you may have done for a 3270 terminal network) to secure access to your applications. The anonymity provided by the Internet, which is so attractive to Web surfers, means that you must consider how to prevent someone from misusing a powerful user ID from the Internet to anonymously achieve a desired objective in your Web-enabled CICS system.

Accessing the transmitted data

The Internet is a public network, and although the risk of the transmitted data being intercepted or modified is low, it is still a distinct possibility. Just as you use registered mail to send valuable parcels or letters, you must take extra precautions when transmitting sensitive data over the Internet. Encryption of the transmitted data using the SSL protocol is a popular solution to prevent access to the transmitted data.

3.1.2 z/OS V1R7 Integrated IP Security as compared to the z/OS Firewall Technologies

Before z/OS 1.7, the IP Filtering and Virtual Private Network (VPN) support was spread among several z/OS components. The management and configuration functions were provided by the z/OS Firewall Technologies, which were part of the z/OS Security Server until z/OS 1.5, and then became part of the so-called Integrated Security Services.

Important: At the time of editing this book, it has been officially announced that the z/OS Firewall Technologies have been discontinued in z/OS V1R8.

Figure 3-1 describes the functional layout of the z/OS Firewall Technologies.

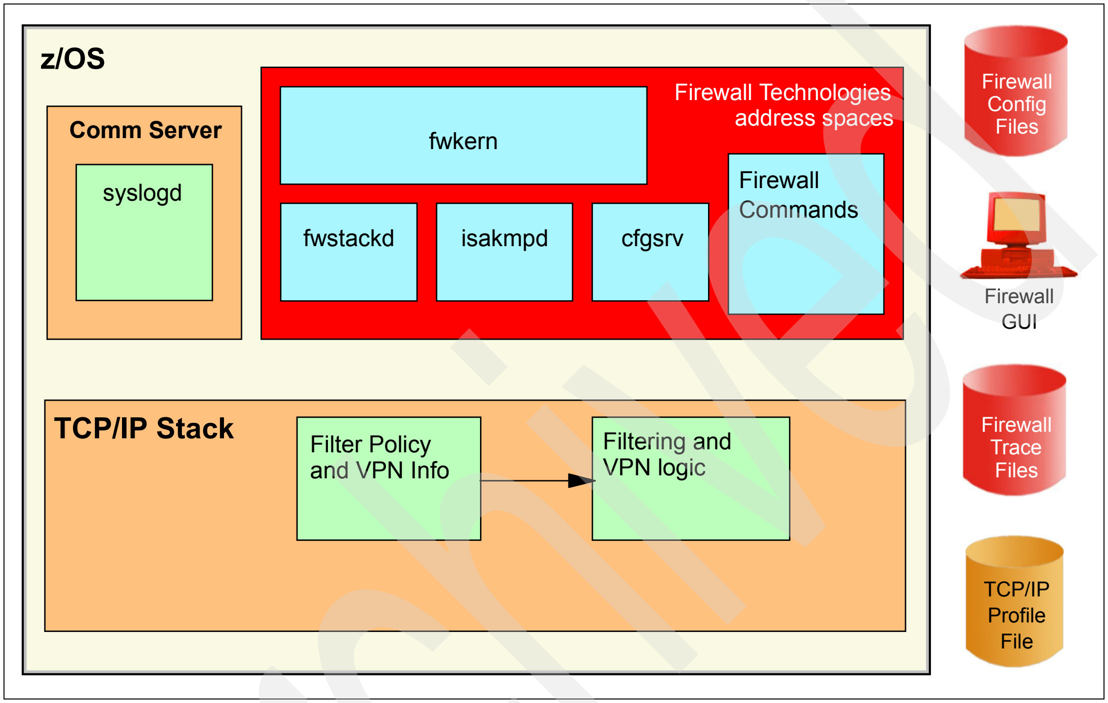


Figure 3-1 IP Filtering and VPN support components in z/OS Firewall Technologies

Compare this to the new IP Security implementation in z/OS V1R7, which is shown in Figure 3-2. Note the higher level of integration, which also shows that the functions are now fully integrated into the z/OS Communications Server package.

Attention: Earlier, the z/OS Firewall Technologies provided other functions in addition to IP Filtering and VPNs. These functions include:

- FTP proxy
- Socks server
- Network Address Translation (NAT)

These functions are no longer available in the new z/OS IP Security functions.

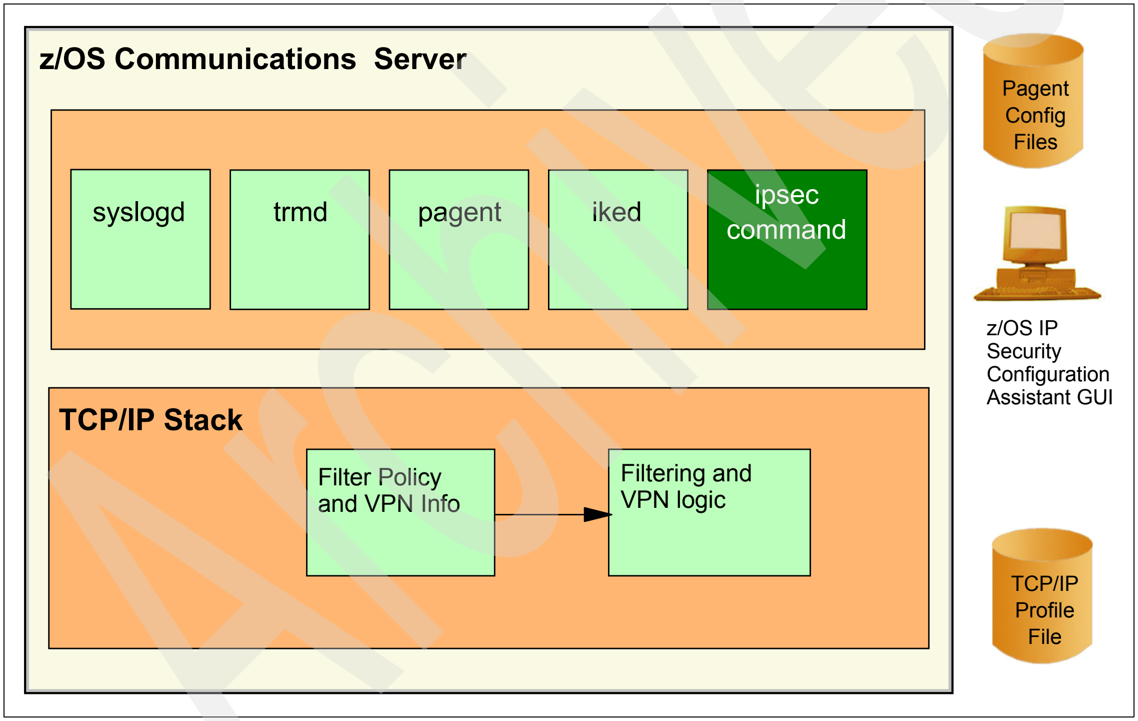


Figure 3-2 New integrated IP Security components in z/OS 1.7

The z/OS V1R7 IP Security services are under the control of the policies installed in the TCP/IP stacks by the Policy Agent utility. The new z/OS UNIX `ipsec` command is used to manage and monitor the IP filtering and VPN policies.

The Communications Server provides integrated functions to support the IP filtering, the IPSec VPNs, and the Internet Key Exchange (IKE) daemon. When compared to the previous Firewall technologies, this implementation provides the following:

- Easier configuration
- Greater scalability
- Improved performance
- Enhanced serviceability

3.2 The z/OS Communications Server Policy Agent

This section discusses the Policy Agent (PAGENT) utility, which is required for setting up the VPN, Application Transparent Transport Layer Security (AT-TLS), IP filtering, and Intrusion Detection Services (IDS). The Policy Agent fetches relevant data in the policies pertaining to these functions and installs this data into the designated TCP/IP stacks in an internal code format.

After the policies are installed in a z/OS stack, the Policy Agent can be invoked with the z/OS UNIX command, **pasearch**. Using the **pasearch** command, a system operator can quickly confirm the status of the policies that are installed in the TCP/IP stacks.

As Figure 3-3 shows, quite a large number of components of a TCP/IP environment are linked to the agent.

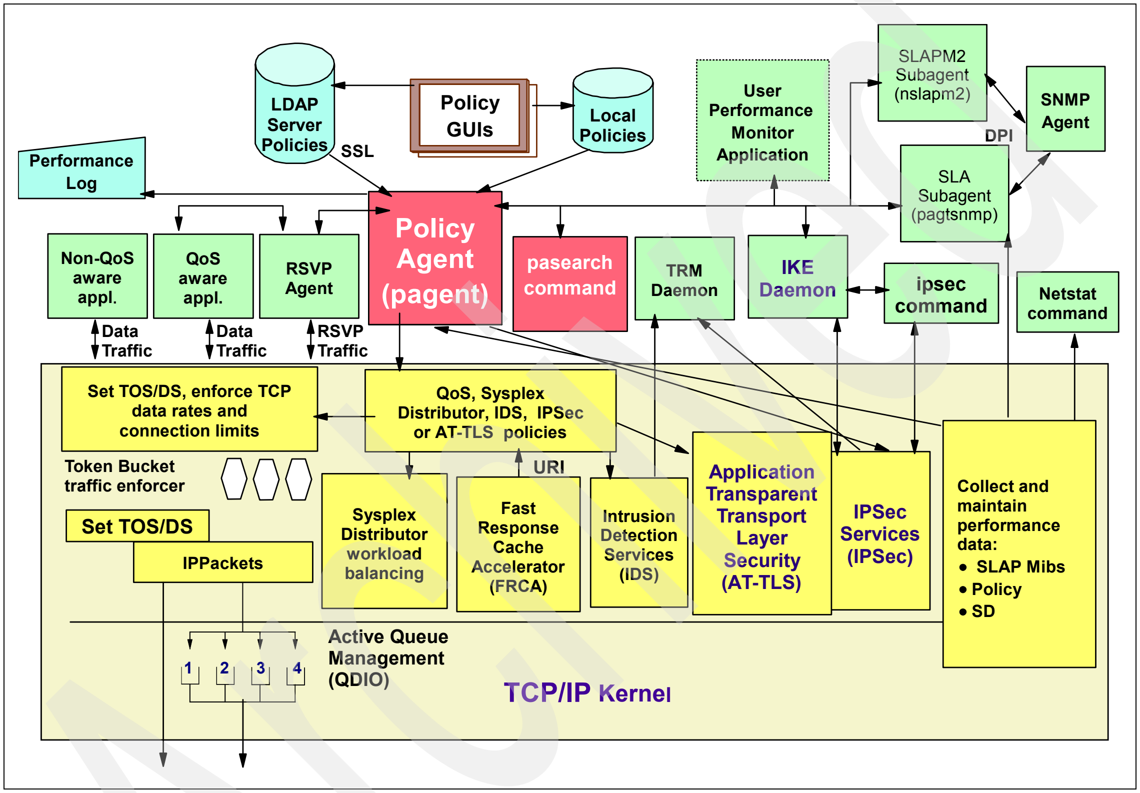

LDAP Server Policies
Policy GUIs
Local Policies
User Performance Monitor Application
SLAPM2 Subagent (nslapm2)
SNMP Agent
DPI
SLA Subagent (pagtsnmp)
Performance Log
SSL
Non-QoS aware appl.
QoS aware appl.
RSVP Agent
Policy Agent (pagent)
pasearch command
TRM Daemon
IKE Daemon
ipsec command
Netstat command
Data Traffic
Data Traffic
RSVP Traffic
Set TOS/DS, enforce TCP data rates and connection limits
QoS, Sysplex Distributor, IDS, IPSec or AT-TLS policies
Token Bucket traffic enforcer
URI
Set TOS/DS
IPPackets
Sysplex Distributor workload balancing
Fast Response Cache Accelerator (FRCA)
Intrusion Detection Services (IDS)
Application Transparent Transport Layer Security (AT-TLS)
IPSec Services (IPSec)
Collect and maintain performance data:
SLAP Mibs
Policy
SD
1
2
3
4
Active Queue Management (QDIO)
TCP/IP Kernel

Figure 3-3 z/OS Communications Server Policy Agent

In Figure 3-3, two sources of policy configuration data are shown above the Policy Agent itself. The Policy Agent can read the data either from an Lightweight Directory Access Protocol (LDAP) server for the IDS and Quality of Service (QoS) policies, or from the flat files for all the policies except the IDS policy, which, even at the time of writing this book, had to be fetched from an LDAP directory.

To the right of the Policy Agent is the **pasearch** command. As mentioned earlier, this command is used to query information about the policies that have been read into a stack by the agent.

Note the box titled IPSec Services. This is the service that implements the IPsec VPN. As mentioned earlier, the VPN is controlled by policy statements. The z/OS UNIX **ipsec** command allows policy information to be queried and changed.

Two other partners of the Policy Agent warrant a comment here. The first is the Traffic Regulation Monitoring daemon (TRMD). This can be viewed simply as a message and report writer, sending the message and the report to Syslogd. TRMD is in charge of event recording for IDS, IPsec services, and traffic regulation. Traffic regulation is a mechanism embedded in z/OS, which limits the amount of TCP or User Datagram Protocol (UDP) requests the stack has to process, thus limiting the effects of denial-of-service attacks.

The z/OS Network Configuration Assistant is a graphical user interface (GUI) that can be used to define AT-TLS and IPsec, VPN, and Internet Key Exchange (IKE) configuration policies. After being defined in the Configuration Assistant, the policy files and other relevant parameters or job control language (JCL) files are sent to the z/OS TCP/IP host using FTP. The GUI can be downloaded from the following URL:

http://www.ibm.com/support/search.wss?tc=SSSN3L&rs=852&rank=&dc=D400&dtm

Figure 3-3 on page 99 also shows the IKE daemon (IKED). As mentioned earlier, IKE stands for Internet Key Exchange, a standardized protocol defined in RFC 2409. The IKE daemon facilitates negotiation and exchange of keys to be used for encryption in a VPN. As such, IKED is really a helper application to make the management of encryption keys for a VPN an automated process.

3.3 Virtual Private Network

Connecting a sysplex member to a nonsecure network introduces the necessity for securing data that travels over the network. A VPN is one of the solutions available on z/OS to improve the security of the data on the wire.

Clarifying the terminology

The world of technology has plenty of confusing acronyms. Therefore, it is only appropriate to make sure that you know the meaning of some of these terms.

A *VPN*, also called a tunnel, is cryptographic protection applied to one or more logical (virtual) connections between one or more pairs of endpoints on an IP network. The traffic between these two endpoints is authenticated and encrypted. The effect is to have data carried in a secure fashion over what is classified as a nonsecure network.

The term *IPsec* (IP Security) is used to describe the standardized protocol that makes a VPN exist and be exploited. The IPsec standard is documented by the Internet Engineering Task Force (IETF) in Request for Comments (RFCs). The IPsec protocol is complex and described over many RFCs. However, the primary document is RFC 2401, which is available on the Web at:

`http://www.ietf.org`

In z/OS V1R7, the replacement mechanisms for the Firewall Technologies fall in the functions category called Communications Server IP Security, and requires the new statement IPSEC in the TCP/IP profile data set to be enabled. This IPSEC statement also controls the IP filtering functions built into the stacks in z/OS V1R7.

In addition, VPN, the IP filtering options, and more are controlled by a new z/OS UNIX command, **`ipsec`**.

In summary:

- IPsec is the public standard for VPN communications
- IPSEC is the statement in the z/OS PROFILE.TCP/IP data set that enables IP filtering or the IPSec VPNs
- **`ipsec`** is a z/OS UNIX command used to configure and display filtering and VPN settings

3.3.1 Internet Security Association and Key Management Protocol and Internet Key Exchange

An IPsec VPN provides encryption, integrity, and authentication, or in other words, privacy and integrity of data and identity verification of the communicating partner. However, keep in mind the fact that with VPNs, the partners or *endpoints* are TCP/IP hosts and not applications as is the case with Secure Sockets Layer/Transport Layer Security (SSL/TLS).

In order to encrypt data, a VPN uses a symmetric encryption key at each endpoint. There are two ways in which this encryption key can be established:

- Manually, by installing the secret keys at both the endpoints
- Dynamically, by using the Internet Security Association and Key Management Protocol (ISAKMP) with the IKE protocol

The manual method has the following limitations:

- The process of sharing the key must be carried out through some secure method such as placing it on a floppy disk and copying it to each endpoint
- Mistakes may occur when typing

- Refreshing the key, a mandatory practice to protect against compromise of the key, must be performed manually
- If the endpoints of the VPN are far apart, securely transporting the key becomes a difficult undertaking.

Although z/OS V1R7 supports manual VPNs, this book does *not* discuss them because dynamic VPNs are considered to consist of a superior method of key exchange.

The ISAKMP and its subsidiary, the IKE, comprise the public standard defined in RFC 2408 andRFC 2409 respectively, for the dynamic setting of a VPN key by providing a secure, automated method of VPN endpoints authentication and exchange of secret encryption keys over a nonsecure network.

3.3.2 Security associations and Virtual Private Network

A security association (SA) is a one-way or simple specification of the cryptographic algorithms, keys, and processes, which the VPN endpoints must adhere to. Because an SA pertains to only one direction of the data flow inside the VPN, two security associations are required to identify the processes that must be run at both ends of the tunnel.

Figure 3-4 is a graphical representation of SAs and their contents.

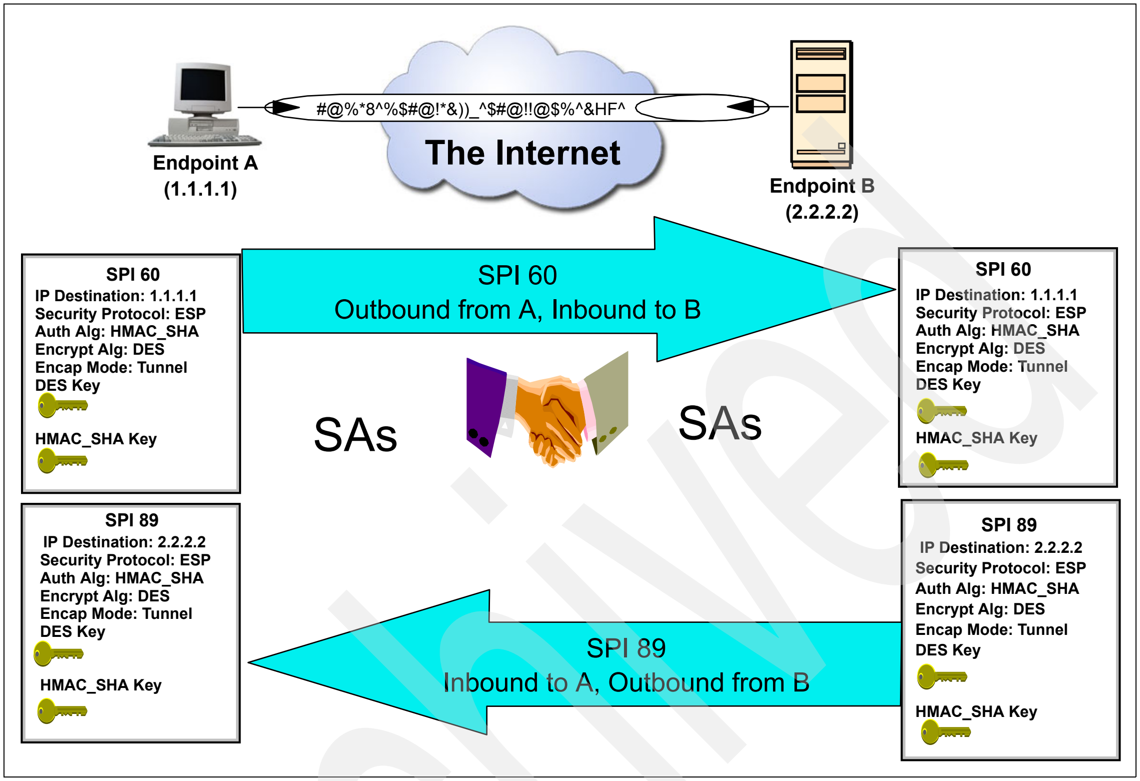


Figure 3-4 Security associations

The SAs are automatically established using the IKE protocols. After being established, SAs are identified by a number called the Security Parameter Index (SPI).

An SA consists of the following information:

- IP destination

  The IP address, inbound and outbound, of the VPN endpoint

- Security protocol

  IPsec allows options such as whether data must be encrypted (Encapsulating Security Payload or ESP) or protected only from modification (Authentication Header or AH). AH does not provide any privacy, the involved cryptographic processes require less resources than ESP protection.

- Authentication algorithm

  The scope of this book does not include a discussion of different authentication algorithms. A successful negotiation ends with both ends using the same algorithm to authenticate.

- Encryption algorithm

  Again, encryption algorithms are outside the scope of this book. A successful negotiation ends with both ends using the same algorithm to encrypt and decrypt.

- Encapsulation mode

  In the *tunnel mode*, the entire IP datagram, header and data, is encapsulated as a new data field and is given a new IP header. The new source and destination IP addresses are the endpoints of the VPN.

  In *transport mode*, the original IP header is left intact, implying that the original source and destination addresses are the VPN endpoints.

- Keys

  The actual encryption keys to be used are also a part of the SA.

RFC 2401 provides the relevant background information pertaining to all these terms.

> **Important:** There are known incompatibilities between the VPN traffic and the Network Address Translation (NAT) function, which, in some cases, leads to the impossibility of properly establishing VPNs if a NAT device is located between the two endpoints. You must be aware of any NAT device that may be traversed by potential tunnels. You must then refer to the VPN implementation reference documentation to determine whether you are facing the impossibility of properly operating the tunnel. For z/OS, the document to refer is *z/OS V1R8.0 Communications Server: IP Configuration Guide*, SC31-8775.
>
> The z/OS V1R7 IPsec VPN technology supports the NAT Traversal protocol, which can help solve some of these cases.

3.3.3 Virtual Private Networks and certificates

The dynamic tunnel method of establishing a VPN uses several options to authenticate both the IKE daemons, that is, both the endpoints of the VPN. One of these options involves an exchange of digital certificates. It is outside the scope of this book to go into details about how certificates work. The digital certificate format is the x.509 V3 format.

Each endpoint sends a certificate request to the other end with information describing the certificate authority certificate that it will accept. In turn, the individual certificate signed by this authority is sent out in response.

3.4 Application Transparent Transport Layer Security

This section describes the AT-TLS implementation provided in z/OS V1R7. With AT-TLS, the z/OS TCP/IP stack can provide TLS support to applications that do not have TLS support implemented in their code.

> **Note:** Secure Sockets Layer (SSL) was developed by Netscape and the current version, V3.0, has been available since 1996. TLS is a secure protocol that supersedes SSL.
>
> TLS is an IETF standard protocol described in RFC 2246, and although based on SSL V3.0, it is not interoperable with SSL. However, experience shows that current products that support TLS can fall back to SSL communication if the partner does not support TLS.

Secure Sockets Layer/Transport Layer Security: Why and how

When an application uses a nonsecure network to send and receive data, some method of protection is recommended. SSL/TLS is a secure client/server protocol that is used by the application to preserve the confidentiality of data with encryption in order to authenticate the server it is connected to, and optionally, to have the authenticating client use a digital certificate.

The SSL/TLS protocol uses well-known encryption and hashing protocols such as:

- Encryption Symmetric Key algorithms: DES, T-DES, RC4, AES
- Hashing algorithms: MD5 and SHA-1
- Encryption Asymmetric Key algorithms: RSA, Diffie-Hellman, and DSA

The protection provided by SSL/TLS differs from the IPSec VPN, in that:

- SSL/TLS is intended for an application-to-application communication, as opposed to a VPN that protects a TCP/IP host-to-host communication
- The SSL/TLS flow appears as normal TCP communication to the network entities. SSL/TLS is not a TCP/IP-registered protocol the way that TCP, UDP, and Internet Control Message Protocol (ICMP) are. IPSec is a registered protocol (protocol 50 and 51) so that network entities know that the packet is transporting IPSec-protected data and can take decisions based on this.

Note that SSL/TLS can flow within a VPN. The VPN ends at the TCP/IP stack network layer and the SSL/TLS communication ends at the application layer.

z/OS V1R7 offers the new AT-TLS function. It is intended for those client/server applications, which do not support SSL/TLS for some reason, for which you want to protect the TCP/IP socket communication with TLS. With AT-TLS, the TLS protocol is handled by the TCP/IP stack.

3.4.1 Application Transparent Transport Layer Security concepts

z/OS has been providing, and still provides the System SSL application programming interface (API) for applications that do not have their own SSL/TLS code, but have to support the protocol. System SSL is widely used by SSL/TLS clients and servers provided with z/OS. System SSL has two specific features:

- It transparently invokes the hardware cryptographic coprocessors, if they are in operation in the system, in order to get hardware assistance for the SSL/TLS cryptographic processes.
- Its API is intended for the C/C++ language

Applications running on z/OS can still have their own SSL/TLS support code. However, if the System SSL API is not being used, it will not get the benefit of the implicit cryptographic coprocessor hardware assist.

With AT-TLS, an application that does not support SSL/TLS or has the support, but is not using the z/OS System SSL API, can transparently get access to System SSL through the TCP/IP stack.

Note that AT-TLS, with its setup parameters, allows a modular involvement of the application with the ongoing TLS communication. The application may completely ignore the fact that TLS is being performed, and is therefore not involved at all (that would be the case for applications without any SSL/TLS support and for applications with their own support that for diverse reasons we do not want to use). Alternately, the application can intervene in the protocol execution sequence to grab, for instance, the digital certificate provided by the partner.

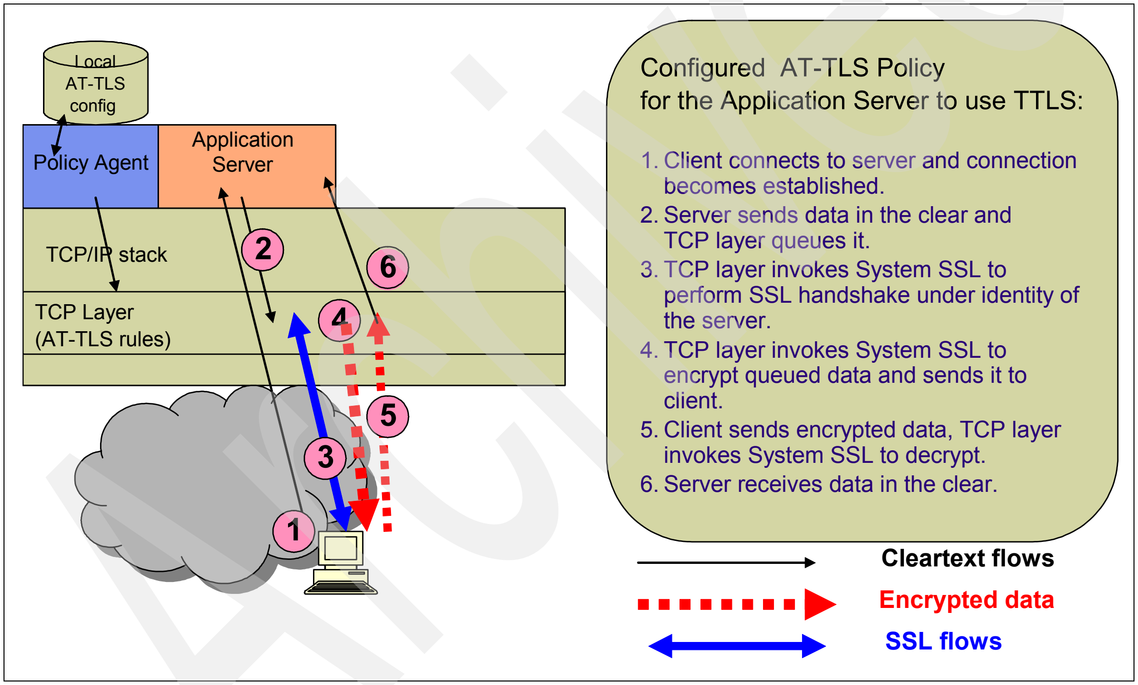
The AT/TLS provides a transparent layer where all the SSL/TLS functions are implemented and defined outside the application in the TCP/IP address space.

3.4.2 Application Transparent Transport Layer Security z/OS implementation

The z/OS V1R7 AT-TLS function involves several components:

- The TCP/IP stack address space
- The PAGENT utility
- System SSL

The TCP/IP address space is responsible for handling all the TLS-related functions such as handshake, authentication, encryption, and decryption of the data, and session key management through calls to System SSL. All the functions are performed based on user-defined policies and are loaded into the TCP/IP stack by the PAGENT. A high-level description of the AT-TLS implementation and data flow is shown in Figure 3-5.



Configured AT-TLS Policy
for the Application Server to use TTLS:

1. Client connects to server and connection becomes established.
2. Server sends data in the clear and TCP layer queues it.
3. TCP layer invokes System SSL to perform SSL handshake under identity of the server.
4. TCP layer invokes System SSL to encrypt queued data and sends it to client.
5. Client sends encrypted data, TCP layer invokes System SSL to decrypt.
6. Server receives data in the clear.

Figure 3-5 AT-TLS flow

In Figure 3-5 the application server does *not* have to know anything about the SSL/TLS protocol. Every function required by the TSL protocol is performed transparently by System SSL on request by the TCP/IP stack address space. For more details about the AT-TLS configuration and implementation, refer to *z/OS V1R8.0 Communications Server: IP Configuration Reference*, SC31-8776, and *z/OS V1R8.0 Communications Server: IP Configuration Guide*, SC31-8775.

The TCP/IP stack address space intercepts any connection being made from the z/OS image or from the network, and takes a decision about whether the connection must use TLS based on the AT-TLS policy.

Note that the client application shown in Figure 3-5 must support TLS. Alternately, it can be a non-TLS-enabled application that runs on z/OS V1R7 and also takes advantage of AT-TLS.

For the sake of security, applications have to make decisions based on the current characteristics of the TLS session and take the appropriate action. This is possible with AT-TLS. The degree of involvement of the applications with the AT-TLS operations is shown in Figure 3-6.

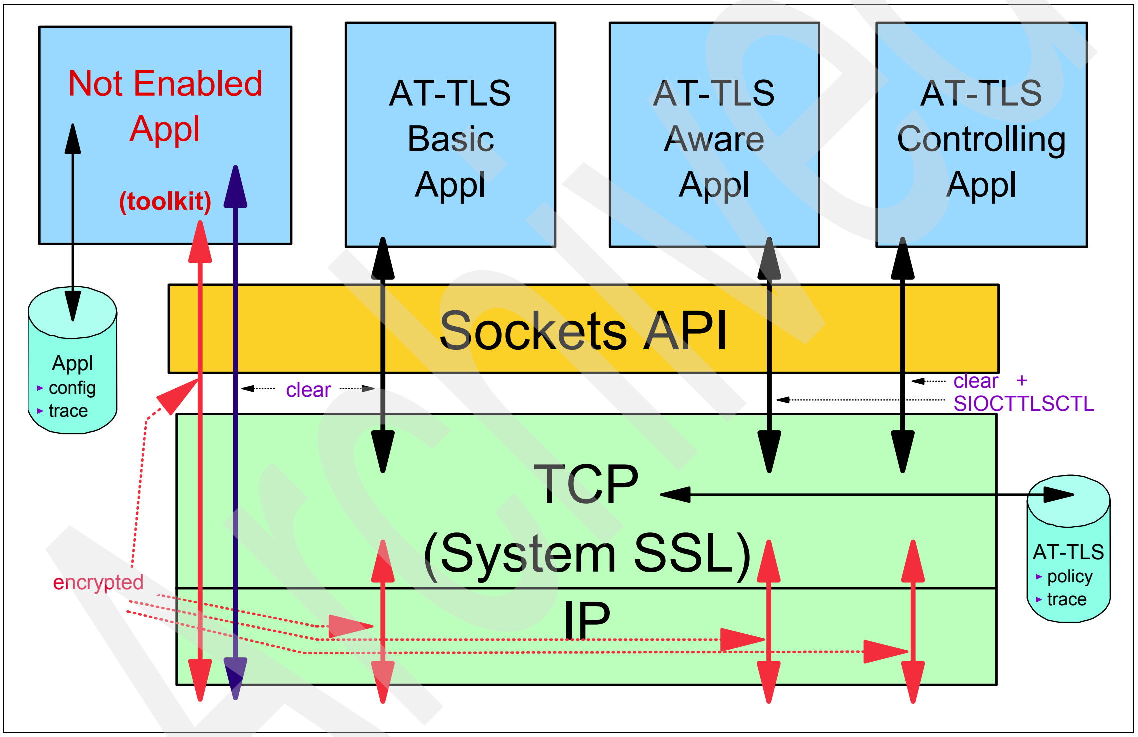


Figure 3-6 AT-TLS application types

Following are the application categories:

- Not-enabled applications, that is, applications that cannot be assisted by AT-TLS. These include:
  - Applications written in Pascal API or Web servers using the Fast Response Cache Accelerator feature

  - No policy is defined to them, or a policy explicitly disables the usage of AT-TLS
  - Applications already using the System SSL API

- Basic applications. These are applications that can be transparently assisted by AT-TLS:
  - There is a policy specifying that they be enabled for AT-TLS usage
  - They are unchanged and are unaware of AT-TLS
  - The application protocol is not affected by the use of AT-TLS
- Aware applications. These are applications that get information from the TLS context and protocol data:
  - There is a policy specifying that they be enabled for AT-TLS usage
  - The application is changed to use a socket API called SIOCTTLSCTL in order to extract information from AT-TLS such as the policy status, the negotiated SSL/TLS version and cipher specifications, and the partner's certificate.
- Controlling applications. These are applications that contribute in some way to the execution of the TLS protocol:
  - There is a policy specifying that they be enabled for AT-TLS usage and for controlling the usage of AT-TLS on the session
  - They can start the session in the clear and later decide to start the SSL/TLS session
  - The SIOCTTLSCTL is used to extract information and to control the AT-TLS, for example, starting a secure session and resetting the cipher or the session

3.5 z/OS intrusion detection services

In this section, we discuss z/OS intrusion detection services.

3.5.1 Overview of intrusion detection

In network security terminology, *intrusion* globally designates anomalous and potentially malicious activities. The objective of an intrusion may be to acquire information that a person is not authorized to have. It may be to gain unauthorized access to a system and use it as a stepping stone for further intrusions elsewhere. Alternately, it may be to cause business harm by rendering a network, system, or application unusable. Most intrusions follow a pattern of information gathering, attempted access, and then destructive attacks.

An intrusion detection system (IDS) can be network-based, in that, it analyzes the data flowing over a segment of the network, or it can be host-based when performing the analysis within the TCP/IP stack of a network host system. It can also be a mix of both technologies, comprising sensors located in network segments and on the host's TCP/IP stacks.

Network-based intrusion detection

Sensors, also called *IDS probes*, analyze the network data against known *signatures*, meaning IP datagram headers or data patterns known to be used for intrusion. Because the probes are scattered over the network, and a single probe view is usually not enough to assess the real danger of the observed IP traffic, network-based IDS also involves the use of correlating devices such as the IBM Tivoli® Risk Manager. These devices raise an alert if the observed traffic is deemed to be a real alarm (in IDS terminology, not a *false-positive*) based on the information collected from several IDS probes.

Host-based intrusion detection

Host-based intrusion detection relies on the additional capabilities on the host TCP/IP stack to analyze the received IP packets against known intrusion characteristic patterns. Host-based IDS presents the following advantages when compared to network-based IDS:

- The ability to evaluate inbound IPSec data from when the data is first decrypted in the target stack before intrusion analysis
- The overhead of per packet evaluation against a table of known attacks is avoided because the target stack can internally detect the anomalous condition and then make a decision based on the IDS policy it has to apply
- Statistical anomalies can be determined based on the target stack internal thresholds or state data
- Prevention methods can be applied by the target stack as per the provided IDS policy
- Globally speaking, there are also fewer false positives with host-based IDS

But again, an installation will probably want to take advantage of both the implementations by integrating network-based and host-based IDS in their network layout.

3.5.2 Understanding the z/OS intrusion detection services

z/OS IDS is an enhanced real-time host-based intrusion detection service that uses policy control to identify, alert, and document suspicious events and assist in their analysis. Traffic regulation is provided for the TCP and UDP traffic through the Traffic Regulation Management Daemon (TRMD). Messages about the possible security violations can be sent to a log file or sent directly to the console. In summary, the z/OS IDS is used for the following:

- Detect and record scanning, common attacks, and flooding
- Record suspicious events to syslog, console, or trace files
- Set up policies to define preventive measures (meaning queue and connection limits) against floods and attacks such as multipacket denial-of-service attacks
- Gather data for possible legal actions

Figure 3-7 shows the z/OS IDS layout in an installation. The IDS is configured with an IDS policy that defines the intrusion events to monitor, along with the actions to take. The IDS policy can only be stored in an LDAP directory. The Policy Agent (PAGENT) reads the policy from the LDAP server. The policy definitions are processed by the Policy Agent and installed in the TCP/IP stack.

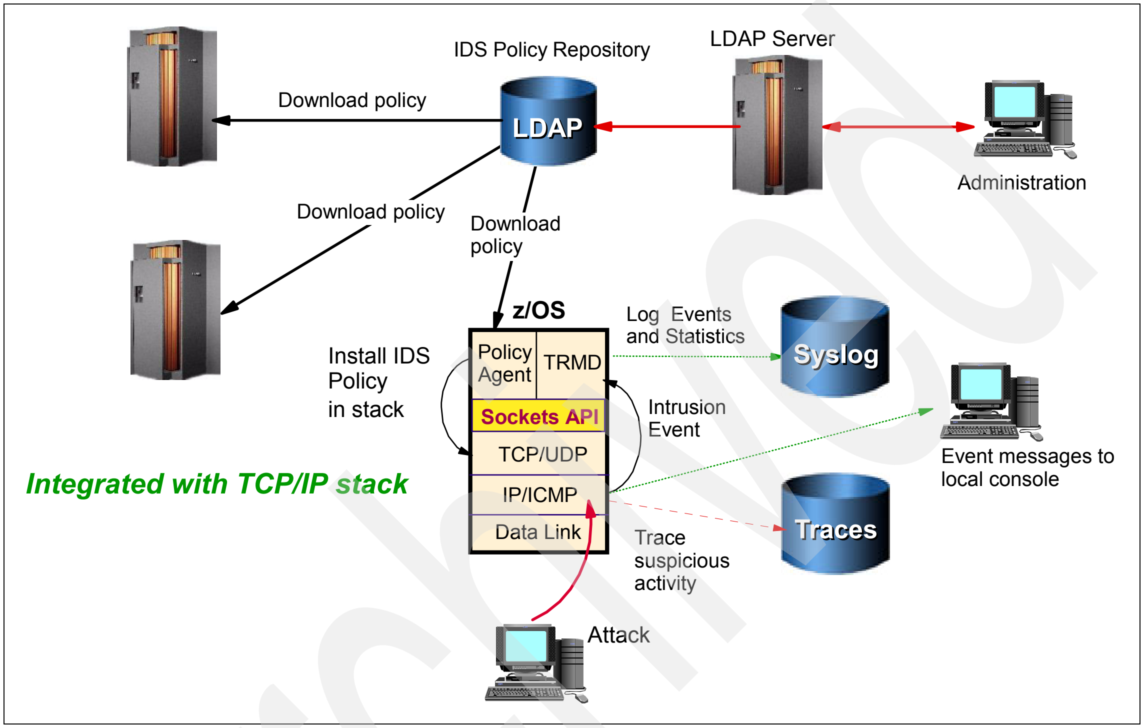


Figure 3-7 Overview of the z/OS IDS infrastructure

3.5.3 The z/OS intrusion detection services policy

The intrusion detection services policy is a set of definitions, entered as attributes in an LDAP directory, which describe to the TCP/IP stacks the event types to monitor, the criteria to apply for issuing alerts, and when to trigger logging and preventive actions.

The supported intrusion event types are described in this section.

Scanning

The intent of scanning is to map the target of the attack by collecting information about subnet structure, addresses, masks, addresses in-use, system type, operating system, application ports available, and software release levels.

Attack

The intent of an attack is to crash or hang the system by sending single malformed packets in an attempt to exploit known weaknesses in the target's system TCP/IP stack or by flooding the target system with multiple packets sent in high volume.

3.5.4 Traffic regulation for TCP connections and UDP receive queues

This is actually a preventive measure in case of a high volume of connection requests arriving at the target host, which could be intended to flood a system or could just be an unexpected peak in valid requests.

It is possible to log suspicious events to the MVS console or syslogd. Statistics are logged to syslogd by the enhanced TRMD. The TRMDSTAT command can be used to summarize and display syslogd messages. It is also possible to trace suspicious packets to the new SYSTCPIS component trace log. Use Interactive Problem Control System (IPCS) to format the SYSTCPIS trace. PAGENT also logs its activity through SyslogD.

Figure 3-8 shows an overview of the interactions between the z/OS IDS components.

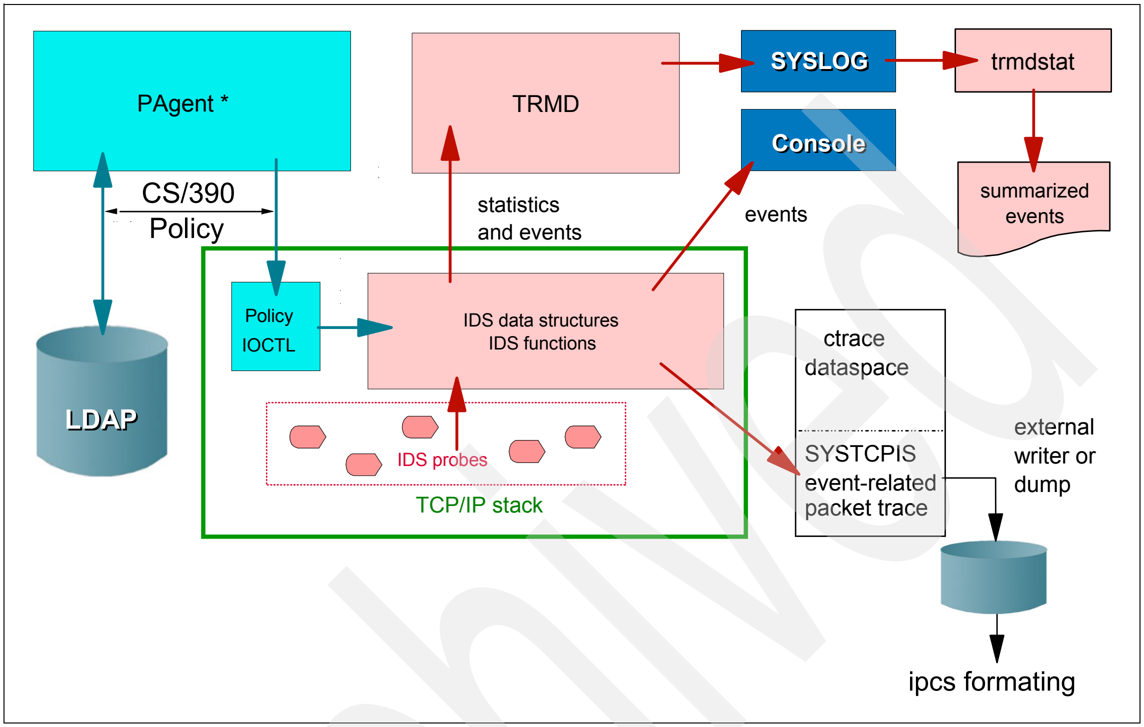


Figure 3-8 Intrusion detection services and IP stack overview

3.5.5 Clarifying the notion of an intrusion detection services event

An intrusion detection services *event* is a countable occurrence of a situation matching a set of conditions defined explicitly or implicitly in the policy. The intrusion detection services event can trigger an action or is a part of the statistics gathered by TRMD, depending on the active intrusion detection services policies. The countable intrusion detection services event is dynamically given a *correlator number* by the TCP/IP stack code that is used to tag the finer detailed information such as traces, which go along with the event.

Example 3-1 shows messages issued by TRMD and displayed in the MVS operator console.

Example 3-1 TRMD messages in the MVS operator console

```
EZZ8762I EVENT TYPE: TCP PORT CONSTRAINED
 EZZ8763I CORRELATOR 3 - PROBEID 01004400
 EZZ8764I SOURCE IP ADDRESS 9.139.240.34 - PORT 0
 EZZ8765I DESTINATION IP ADDRESS 0.0.0.0 - PORT 80
 EZZ8766I IDS RULE trtcpHttp-rule
 EZZ8767I IDS ACTION trtcpHttpLog-action
 EZZ8761I IDS EVENT DETECTED 544
```

Example 3-2 shows messages received and stored by SyslogD.

Example 3-2 TRMD SyslogD stored messages

```
EZZ9321I TRMD TCP constrained entry:02/01/2002
21:16:37.18,lhost=0.0.0.0,port=80,host=9.139.240.34,available=0,total=1
,percent=50,correlator=3,probeid=01004400,threshold=0
```

3.5.6 Traffic Regulation Management Daemon

The Traffic Regulation Management Daemon manages the IDS message collection. It can be run as a started task or can be started from the USS shell. There must be one TRMD instance per TCP/IP stack in the z/OS image. The affinity between the TRMD instance and the stack is indicated in the RESOLVER_CONFIG environment variable.

3.6 Traditional CICS security

In a CICS environment, the assets you normally want to protect are the application programs and the resources that are accessed by the application programs. To prevent disclosure, destruction, or corruption of these assets, control the access to the CICS region and to the different CICS components.

You can limit the activities of a CICS user to only those functions the user is authorized to use, by implementing one or more of the following CICS security mechanisms:

- Transaction security

  This ensures that users who attempt to run a transaction are entitled to do so.

- Resource security

  This ensures that users who use CICS resources such as files and transient data queues, are entitled to do so.

- Command security

  This ensures that users who use CICS system programming commands are entitled to do so.

- Surrogate security

  This ensures that a *surrogate* user is authorized to act on behalf of another user.

When CICS security is active, requests to attach transactions and requests by transactions to access resources are associated with a user ID. When a user makes a request, CICS calls the external security manager, for example, RACF, to determine if the user ID has the authority to make the request. If the user ID does not have the correct authority, CICS denies the request.

In many cases, a user is a human operator, interacting with CICS through a terminal or a workstation. However, the user can also be a Web browser user or, in a Web services solution, a program executing on a client system.

For more details about CICS security, refer to *CICS Transaction Server for z/OS V2.3 RACF Security Guide*, SC34-6249.

3.6.1 CICS user IDs

When a human operator signs in to a CICS region at the start of a terminal session, the person is asked to provide a user ID and a password. The user ID remains associated with the terminal until the terminal operator signs out. Transactions executed from the terminal and requests made by those transactions are associated with that user ID.

For connections from Web users, there are other ways in which the user of a CICS transaction can be identified. This includes the following:

- A Web browser user is challenged to provide Hypertext Transfer Protocol (HTTP) basic authentication information (a user ID and a password). The transaction that services the client's request and other requests made by that transaction are associated with that user ID.

- A client program that is communicating with CICS using the SSL supplies a client certificate to identify itself. The security manager maps the certificate to a user ID. The transaction that services the client's request and other requests made by that transaction are associated with that user ID.

In addition to these transport-level authentication mechanisms, Web service clients may also pass authentication data in the SOAP message.

CICS special user IDs

There are two particular user IDs that CICS uses in addition to those that identify individual users. These are:

- Region user ID

  The CICS region user ID is used for checking the authorization when the CICS system, rather than an individual user of the system, requests access to system resources such as CICS data sets and other servers.

- Default user ID

  When a user does not sign in, CICS assigns a default user ID to the user. It is specified in the system initialization table (SIT) parameter DFLTUSER. In the absence of more explicit identification, it is used to identify TCP/IP clients that connect to CICS. Provide very little authority to the default user ID.

Part 2

Designing the secure CICS SOA solution

This part of the book looks at the different technologies individually, and discusses how best to secure these in an SOA environment.

4

CICS Web services

CICS Transaction Server (TS) V3.1 allows a CICS application to participate in a Web services environment as a service requester or as a service provider. This service-oriented architecture (SOA) environment uses SOAP messages in Extensible Markup Language (XML) to transmit information through a network to a Web services partner. CICS provides outbound support (in which case, CICS is the service requester) and inbound support (in which case, CICS is the service provider). A CICS application can use SOAP messages to communicate with any partner that can understand SOAP, regardless of the platform or the location of the partner. How you secure CICS Web services in this open environment involves understanding not only your local environment, but the pathways to all the other Web service partners with which CICS may communicate.

This chapter discusses the industry standards for Web services and describes the various security options available. It then discusses why you can select one over another or combine several alternatives to meet the requirements of your environment. Following are the topics that are discussed:

4.1 CICS TS and external standards

CICS TS support for Web services conforms to a number of standards that are highlighted in the following sections. The standards and specifications are published by the following organizations:

- World Wide Web Consortium (W3C)
- IBM
- Web Services Interoperability Organization (WS-I)
- Organization for the Advancement of Structured Information Standards (OASIS).

4.1.1 Web services and service-oriented architectures

An SOA has been used under various guises for many years. It can be (and has been) implemented using a number of different distributed computing technologies. The effectiveness of service-oriented architectures in the past has always been limited by the ability of the underlying technology to interoperate across the enterprise.

Web services technology is the ideal technology choice for implementing an SOA. For the first time, all the major vendors are recognizing and providing support for Web services. WS-I is an organization that promotes open interoperability between Web services regardless of the platforms, the operating systems, or the programming languages used.

4.1.2 CICS TS Web services and industry standards

CICS TS V3.1 support for Web services conforms to a number of industry standards and specifications, including the following:

- Extensible Markup Language (XML) V1.0

  http://www.w3.org/TR/REC-xml

- SOAP V1.1 and SOAP V1.2

  - http://www.w3.org/TR/soap12-part1/
  - http://www.w3.org/TR/soap12-part2/
  - http://www.w3.org/TR/soap12-part0/

> **Note:** The SOAP V1.2 Part 0: Primer states that SOAP 1.2 will *not* spell out the acronym for SOAP. Therefore, the complete name is SOAP V1.2.

- Web Services Description Language (WSDL) V1.1

  http://www.w3.org/TR/wsdl
- Web Services Coordination (WS-Coordination) V1.0

  http://www.ibm.com/developerworks/library/specification/ws-tx/
- Web Services Atomic Transaction (WS-Atomic Transaction) V1.0

  http://www.ibm.com/developerworks/library/specification/ws-tx/
- WS-I Basic Profile (WS-I BP) V1.1

  http://www.ws-i.org/Profiles/BasicProfile-1.1.html
- WS-I Simple SOAP Binding Profile (SSBP) V1.0

  http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html
- Web Services Security (WSS): SOAP Message Security

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
- Web Services Security (WSS): Username Token Profile V1.0

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf
- Web Services Security (WSS): X.509 Certificate Token Profile V1.0

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf
- XML Encryption Syntax and Processing

  http://www.w3.org/TR/xmlenc-core
- XML Signature Syntax and Processing

  http://www.w3.org/TR/xmldsig-core
- Hypertext Transfer Protocol (HTTP) V1.1

  http://www.w3.org/Protocols/rfc2616/rfc2616.html

4.1.3 CICS compliance with Web service standards

CICS is compliant with the supported Web services standards and specifications, in that, it allows you to generate and deploy Web service applications that are compliant.

> **Note:** CICS does *not* enforce the compliancy of the Web services that you generate and deploy, for example, CICS allows you to apply additional qualities of service to your Web service that could break the interoperability outlined in the WS-I Basic Profile V1.1 specification.

WS-I Basic Profile V1.1 and CICS compliance

Web services can be used to connect computer systems together across organizational boundaries. Defining a set of open, nonproprietary standards to which all Web services adhere maximizes the ability to connect disparate systems together.

WS-I has released a *basic profile* that outlines a set of specifications to which WSDL documents and SOAP messages sent over HTTP must adhere to in order to be WS-I compliant. The complete list of specifications is available on the WS-I Web site:

http://www.ws-i.org/

CICS TS V3.1 support for Web services ensures maximum interoperability with other Web service implementations by conforming to the WS-I Basic Profile 1.1 and the WS-I Simple SOAP Binding Profile 1.0. Conforming to both the profiles is equivalent to conforming to the WS-I Basic Profile 1.0. For more information about these profiles, refer to:

- WS-I Basic Profile 1.1

  http://www.ws-i.org/Profiles/BasicProfile-1.1.html

- WS-I Simple SOAP Binding Profile 1.0

  http://www.ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html

CICS conditionally complies with the WS-I Basic Profile 1.1 in that it adheres to all the *MUST* level requirements. CICS does not specifically implement support for UDDI registries, and the points relating to this in the specification are ignored. Also, the CICS Web services assistant batch utilities (DFHLS2WS and DFHWS2LS) and the associated runtime environment are not fully compliant with this profile because there are limitations in the support for mapping certain schema elements. For more information, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

4.2 Web services security exposures

Web services security requires that you understand the potential security exposures when using Web services, and the methods available to secure the environment. This section first discusses the security exposures that you may face, and later discusses the options available to you for creating a secure framework for your data. The transport-level security and Web Services-Security (message-level security) sections provide you with an overview of each of them and discuss the pros and cons associated with each method. This provides you with the information you require to determine which method or combination of methods to implement in your environment.

Web services security is one of the most important Web services subjects. When using Web services, similar security exposures exist as for other Internet, middleware-based applications, and communications. To explain the Web services security exposures, a bank teller scenario is used, as shown in Figure 4-1. The bank teller (Web service requester or client) connects over the Internet to the bank's data center (Web service provider). Assume that *no* security is applied, which, although not realistic, is used for this example.

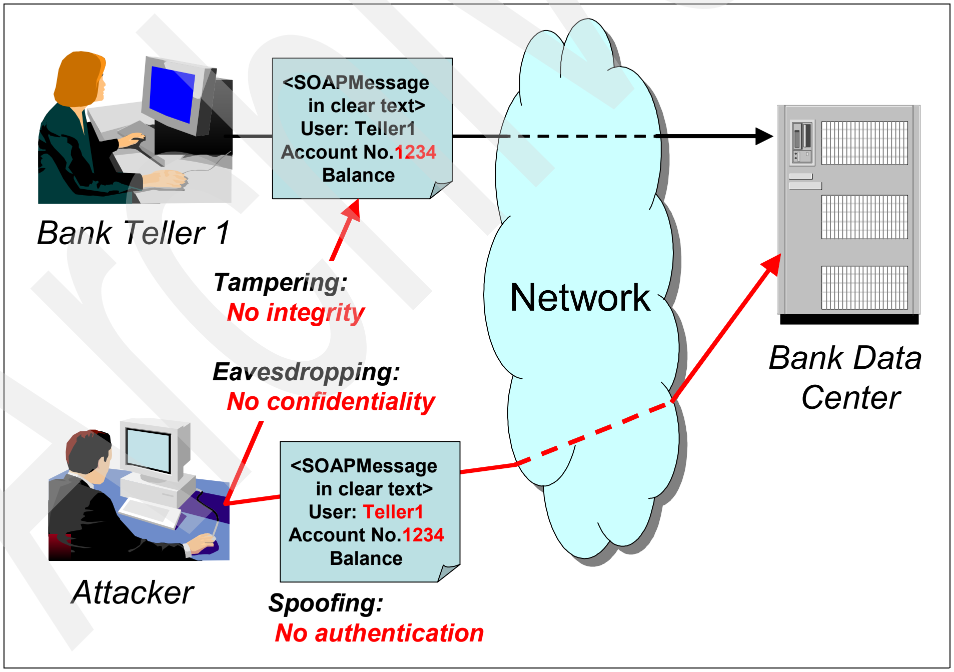


Figure 4-1 Potential security exposures in a Web services environment

The three major risk factors in this example are:

- Spoofing, no authentication

  An attacker, posing as a bank teller, can send a modified SOAP message to the service provider IN ORDER to get confidential information or to withdraw money from another client's account.

  Eliminate this security exposure by applying authentication to the Web service.

- Tampering, no integrity

  The SOAP message is intercepted between the Web service client and the server. An attacker can modify the message by changing the account number and cause money to be deposited into another account. Because there is no integrity constraint, the Bank Data Center Web service does not check if the message is valid, and accepts the modified transaction.

  Eliminate this security exposure by applying an integrity mechanism to the Web service.

- Eavesdropping, no confidentiality

  An attacker can intercept the SOAP message and read the information contained in the message. Because the message is not encrypted, confidential client information or bank information can be viewed by the attacker. This exposure exists because the account number and balance information is sent over the network in plain text.

  Eliminate this security exposure by applying a confidentiality mechanism to the Web service.

To prevent these security exposures, apply the following mechanisms to secure a Web services environment (Figure 4-2):

- Transport-level security: Transport Layer Security (TLS) or Secure Sockets Layer (SSL)
- Message-level security: Web Services Security (WS-Security)

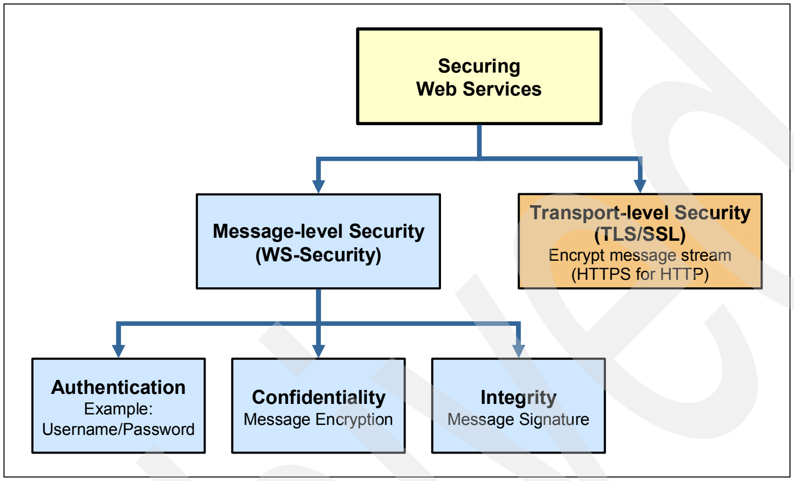


Figure 4-2 Securing Web services

Depending on the level of application security you require, apply one or more of these security mechanisms. Also, depending on other nonfunctional requirements such as system capacity and performance, you can implement a combination of transport-level security and message-level security.

The more the security mechanisms you implement, the more potential you have to negatively affect your other nonfunctional requirements. Therefore, when designing a Web services security solution, keep in mind that security has an impact on the following:

- System capacity

  Any applied security mechanism has an impact on system resource usage, for example, CPU and memory usage. Therefore, when planning a Web service environment, the required extra processing for security must be considered in system capacity and volume planning.

  The nonfunctional requirements, capacity and volume, cover the number of concurrent users and the number of transactions per second. This influences the required system infrastructure, including hardware, software, and the network.

- Performance

  Security mechanisms and functions also impact the application’s response time. Keep this in mind when defining the Web service system response time requirements.

  The performance requirement for your system may be defined by the total response time for your main applications, for example, you may have a response time requirement of less than one second for 90% of all transactions.

> **Note:** Applying security is not just a question of feasibility. The additional system resources required and the influence on the response time must also be considered.

4.3 Transport-level security

Web services messaging relies on two protocol layers, the transport layer and the SOAP message layer. Security can be implemented within either or both these layers.

This section reviews how to use the different transport-level security mechanisms to secure a CICS Web services solution. Message-level security mechanisms are discussed in 4.4, “Web Services-Security” on page 137.

4.3.1 Hypertext Transfer Protocol basic authentication

When a CICS Web service is invoked using Hypertext Transfer Protocol (HTTP), use the standard HTTP security mechanisms to authenticate the Web service client and to ensure message integrity and confidentiality.

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and a password) from a client. The client passes the authentication information to the server in an HTTP authorization header.

The AUTHENTICATE attribute in the CICS TCPIPSERVICE resource definition specifies the authentication and identification scheme to be used for inbound TCP/IP connections for the HTTP protocol. HTTP basic authentication is enabled with CICS by specifying BASIC for the AUTHENTICATE attribute. For more information about the TCPIPSERVICE parameters and attributes, refer to *CICS Transaction Server for z/OS V3.1 CICS Resource Definition Guide*, SC34-6430.

A CICS service provider application can be protected by HTTP basic authentication. However, the HTTP basic authentication scheme can only be considered a secure means of authentication if the connection between the Web service client and the CICS region is secure. If the connection is insecure, the scheme does *not* provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication must be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

4.3.2 Secure Sockets Layer or Transport Layer Security with Hypertext Transfer Protocol

SSL or TLS is a popular way of encrypting communication between business partners over the Internet. The TLS 1.0 protocol is the latest industry standard SSL protocol and is based on SSL 3.0. The TLS 1.0 specification is documented in Request for Comment (RFC) 2246. For more information about RFC 2246, refer to the following RFC index search engine at:

http://www.rfc-editor.org/rfcsearch.html

SSL or TLS provides transport-level protection. It creates a secure connection between two nodes and encrypts all the traffic flowing between the nodes. This is also referred to as *point-to-point* security.

Because most Web services flow over HTTP today, SSL or TLS provides a straightforward way of ensuring confidentiality. It also includes a built-in communication integrity check. Connection layer authentication is achieved by the client always authenticating the server, and optionally being authenticated by the server through the exchange of X.509 certificates.

The client initiates a Hypertext Transfer Protocol Secure (HTTPS) connection by using an Uniform Resource Locator (URL) starting with https: instead of http:. With SSL or TLS, the data flowing back and forth between the client and the server is encrypted using a secret key algorithm. The exchange of the secret key occurs at the start of the communication during the *SSL handshake*. This is illustrated in Figure 4-3.

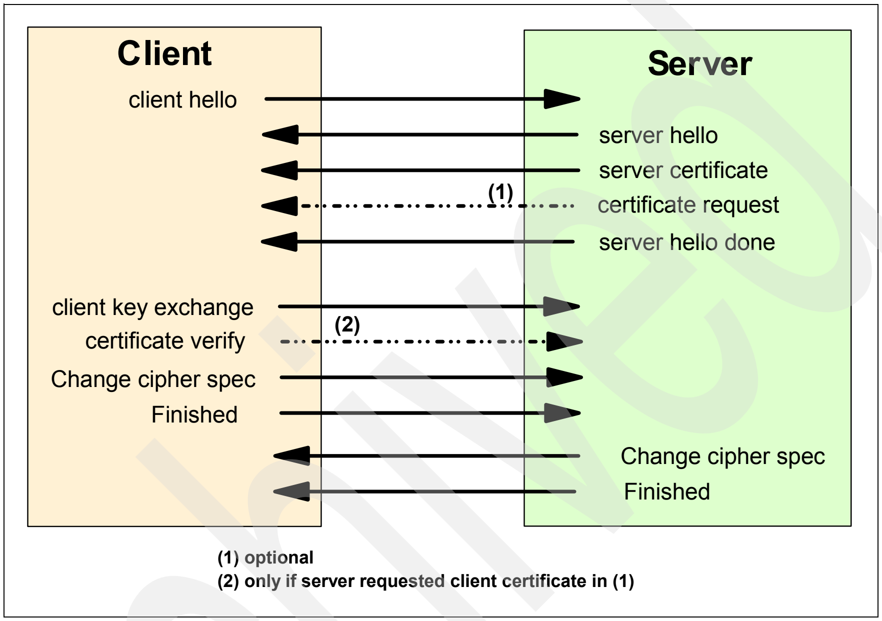


Figure 4-3 SSL handshake

Public key encryption is a cryptographic system that uses two keys, a *public key* that is potentially known to everyone, and a related *private key* that is known only to one party in an exchange of information. For more information, refer to *3.4.2, "Application Transparent Transport Layer Security z/OS implementation" on page 107* of this book, and *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454.

The server's certificate contains the server's public key. The client uses this public key to encrypt an initial value for a secret key to be sent to the server. After the client and the server have obtained the same secret key value, the handshake ends, and they change their encryption algorithm to the new and less CPU-intensive secret key procedure. The server can also request a digital certificate from the client in order to verify its identity.

4.3.3 CICS support for Secure Sockets Layer orTransport Layer Security

CICS supports the SSL 3.0 and the TLS 1.0 protocols. HTTPS connections automatically use the TLS protocol, unless the client specifically requires SSL 3.0.

A CICS service provider application can be secured using HTTPS, which has the following advantages:

- It can be used to provide a fast and secure transport for CICS Web services.
- It provides for authentication through either HTTP basic authentication or a client X.509 certificate.
- It provides integrity between the service requester and CICS by using symmetric key cryptography to establish the authenticity of the server and the client and to securely share a secret key. Symmetric key algorithms use the same or similarly related cryptographic keys for encryption and decryption.
- It provides confidentiality between the service requester and CICS through efficient shared key cryptography.
- It can be used with hardware cryptographic devices, which can significantly reduce the cost of SSL handshakes. You can customize your encryption settings to use only the cipher suites that use the Integrated Cryptographic Facility (ICSF) such as the Data Encryption Standard (DES) and Secure Hash algorithm-1 (SHA-1) cipher suites. For more information about cipher suites, refer to 4.3.4, “Cipher suites” on page 132.
- It is mature and similarly implemented by most vendors. It is, therefore, subject to a few interoperability problems.

A CICS service requester application can also use HTTPS to invoke a service provider application, with advantages that are similar to those listed here.

There are several CICS System Initialization Table (SIT) parameters that are used to enable CICS support for SSL or TLS.

- ENCRYPTION
- KEYRING
- MAXSSLTCBS
- SSLDELAY
- SSLCACHE

> **Note:** The CRLPROFILE system initialization table parameter may also be of interest to you. It is used to specify the profile name for certification revocation lists (CRLs).

For more information about the system initialization table parameters listed earlier and other parameters that affect CICS security, refer to *CICS Transaction Server for z/OS V3.1 CICS System Definition Guide*, SC34-6428.

The SSL attribute of the TCPIPSERVICE definition is used to activate SSL in CICS. If you require SSL, set it to either `YES` or `CLIENTAUTH`. The AUTHENTICATE attribute of the TCPIPSERVICE definition is used to specify the authentication and identification scheme used for inbound TCP/IP connections for the HTTP and USER protocols, and Internet Inter-ORB Protocol (IIOP).

For more information about enabling CICS support for SSL/TLS and activating SSL for a TCPIPSERVICE, refer to *Implementing CICS Web Services*, SG24-7206.

For more information about the TCPIPSERVICE parameters and attributes, refer to *CICS Transaction Server for z/OS V3.1 CICS Resource Definition Guide*, SC34-6430.

4.3.4 Cipher suites

A cipher represents a type of algorithm used to encrypt and decrypt data. Cipher suites are sets of ciphers that are used in an SSL connection. When an SSL connection is established, the client and the server exchange information about which cipher suites they have in common. The client and the server then communicate using the highest level of security that they both support. If there is no common cipher suite, the SSL handshake fails and the connection is closed because a secure connection is not possible.

The cipher suites must be supported by the operating system in order for CICS to use them. The list of cipher suites supported by z/OS depends on the release number of your operating system. Table 4-1 and the example that follows are for z/OS 1.4 and CICS TS V3.1. Additional cipher suites may be available depending on the release of your operating system. Table 4-1 shows an example of z/OS 1.6 ciphers.

Table 4-1 Cipher suites

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA
03	RC4	40 bits	MD5
04	RC4	128 bits	MD5

Cipher suite	Encryption algorithm	Key length	MAC algorithm
05	RC4	128 bits	SHA
06	RC2	40 bits	MD5
09	DES	56 bits	SHA
0A	Triple DES	168 bits	SHA
2F	AES	128 bits	SHA
35	AES	256 bits	SHA

Following are the terms used in Table 4-1:

- MD5: Message Digest algorithm
- SHA: Secure Hash algorithm
- RC2, RC4: Rivest encryption
- DES: Data Encryption Standard
- Triple DES: DES applied three times
- AES: Advanced Encryption Standard

The CICS ENCRYPTION system initialization table parameter specifies the level of encryption that CICS must use. Beginning with CICS TS V3.1, the default for the ENCRYPTION parameter is STRONG. This means that CICS can use all the cipher suites shown earlier (and additional ones depending on the release number of the operating system) to negotiate with clients.

Using CIPHERS to set a minimum level and a maximum level

You can set a minimum as well as a maximum encryption level by editing the list of cipher suites in the CIPHERS attribute in the appropriate CICS resource definition:

- URIMAP: For outbound HTTP and Web service requests
- TCPIPSERVICE: For inbound HTTP and IIOP
- CORBASERVER: For outbound IIOP

Note: You can also specify CIPHERS on the EXEC WEB OPEN command for outbound HTTP connections.

Setting a minimum encryption level provides you with more control over the security of the SSL connections that you will allow. The following list, for example, provides the list of cipher suite codes for each level of the ENCRYPTION attribute:

- WEAK: Default value is 03060102
- MEDIUM: Initial value is 0903060102
- STRONG: Initial value is 0504352F0A0903060102

This list shows the cipher suites available with z/OS 1.4 and CICS TS V3.1. You can have additional cipher suites available, depending on the release number of your operating system:

If you do not want SSL to allow the lesser cipher suite codes during the SSL negotiation with the partner, set a minimum level by editing the CIPHERS attribute to remove the cipher suite codes that you do not allow. If you want to, for example, enforce STRONG and not allow the negotiation to drop to WEAK, remove the MEDIUM and WEAK codes and keep the unique STRONG codes. In this case, your CIPHERS attribute must have 0504352F0A. If the partner does not support the higher level security codes, the SSL handshake fails and the connection is not established with the partner.

Important: If the SSL handshake negotiates down to using cipher suite 01 or 02, there is no encryption, and data will be transmitted in the clear. It is recommended that you remove 01 and 02 from the list of ciphers if even a minimum amount of encryption is required.

4.3.5 Setting the user ID on the URIMAP

URIMAP definitions are CICS resource definitions that match the Uniform Resource Identifiers (URIs) of HTTP or Web service requests, and provide information about how to process the request. PIPELINE definitions are CICS resource definitions that provide processing information to be performed on the SOAP message. The URIMAP identifies the PIPELINE resource to associate with the URI that was matched for the request, and the PIPELINE specifies the processing that is to be performed on the message.

You can specify a user ID on the URIMAP to be used on behalf of a Web service client. To do this, set the USERID attribute of the URIMAP definition for a request. The USERID attribute specifies the 1 - 8 character user ID under which the Web services pipeline alias transaction will be attached. By specifying the USERID attribute, you can control the security within CICS for the alias transaction. You can also use the TRANSACTION attribute of the URIMAP to specify your name for the alias transaction.

> **Note:** If you define and install URIMAP resource definitions explicitly using Resource Definition Online Transaction (CEDA), and you are using the CICS Web services assistant batch utilities (DFHLS2WS or DFHWS2LS) in order to create the Web services, you cannot take advantage of the dynamic installation of URIMAP resources when the PIPELINE resource is installed. This is because the USERID parameter can also be set in the CICS Web services assistant batch utilities, in which case, it will be used to define the USERID attribute of the URIMAP resource when it is created automatically during the PIPELINE scan. If you explicitly code USERID for the URIMAP and install the URIMAP, it overrides the dynamic installation of the URIMAP resource when the PIPELINE resource is installed.

A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained directly from the client.

> **Important:** If you use a URIMAP definition to set a user ID, there is no authentication of the client's identity. You should do this only when communicating with your own client system, which has already authenticated the user, and communicates with the server in a secure environment.

For more information about the USERID attribute for the URIMAP resource definition, refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458, and *CICS Transaction Server for z/OS V3.1 CICS Resource Definition Guide*, SC34-6430.

Additional URIMAP parameters

Following is a list of the additional URIMAP parameters:

- USAGE

  This determines what other attributes in the URIMAP definition can be used. It can be set to either SERVER (if CICS is the HTTP server), CLIENT (if CICS is the HTTP client), or PIPELINE (for a Web service).

- SCHEME

  This specifies either HTTP without SSL or HTTP with SSL. This attribute is for all USAGE options. If SCHEME(HTTPS) is specified, CICS checks during install time whether SSL is active in the CICS region. The CICS KEYRING system initialization table parameter must specify the key ring used by the CICS region for SSL to be active. If SSL is not active in the CICS region, the URIMAP definition is not installed and the message DFHAM4905 is issued.

- CIPHERS

  This specifies a string of 2-digit cipher suite codes to be used during the SSL handshake. This attribute is for USAGE(CLIENT) only. CICS validates the list of cipher suites against the cipher suites supported on the running operating system. If no valid cipher suites are found in the list, CICS issues the message DFHAM4918, and the URIMAP definition is not installed. If some of the cipher suites in the list are supported, CICS issues the message DFHAM4917, and the URIMAP is installed with the reduced set of cipher suites.

- CERTIFICATE

  This specifies the label of the X.509 certificate to be used as the SSL client certificate during the SSL handshake. This attribute is for USAGE(CLIENT) only. CICS validates the certificate against those specified in the key ring. If the specified certificate is not valid, CICS issues messages DFHAM4889 and DFHAM4928, and the URIMAP definition is not installed.

- TRANSACTION

  This specifies the name of the alias transaction to be used to run the user application that composes the HTTP response, or to start the pipeline. If you specify your own transaction name, it allows you to control security at the transaction level. You may also want to use a unique transaction name for monitoring and accounting or for transaction class (tclass) limits. This attribute is for USAGE(SERVER) and USAGE(PIPELINE) only.

- USERID

  This specifies the user ID under which the alias transaction is attached. You can control the security for the alias transaction based on the user ID specified. If surrogate user checking is enabled in the CICS region (XUSER=YES as a system initialization table parameter), CICS checks that the user ID used to install the URIMAP definition is authorized as a surrogate of the user ID specified for the USERID attribute. This attribute is for USAGE(SERVER) and USAGE(PIPELINE) only.

4.3.6 Determining the user ID order of precedence when using HTTP

It is possible that for a single Web service request transported by HTTP, multiple methods for setting the user ID will be used at the same time. In this event, the following order of precedence is used for setting the user ID under which the target business logic program runs:

1. A user ID specified by a message handler or a SOAP header processing program, which is included in the pipeline that processes the SOAP message. A SOAP header processing program can, for example, extract a user name from the SOAP message and specify that the CICS task must run with this

user ID. For more information about message handlers and SOAP header processing programs, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

2. A user ID obtained from the Web client using basic authentication or a user ID associated with a client certificate
3. A user ID specified in the URIMAP definition for the request
4. The CICS default user ID, if no other ID can be determined

4.4 Web Services-Security

The Web Services Security (WS-Security) specification provides message-level security, which is used when building secure Web services to implement message content integrity and confidentiality.

WS-Security provides privacy and integrity protection by digitally signing and encrypting XML elements in a SOAP message. The body of the message, any of the elements within the body, or the header can be protected. Different levels of protection can be provided to different elements within the SOAP message.

The advantage of using WS-Security over SSL is that it can provide *end-to-end* message-level security. This means that message security can be protected even if the message goes through multiple services called intermediaries. SSL security is considered to be *point-to-point*, and the data may be decrypted before reaching the intended recipient.

As illustrated in Figure 4-4, if the service requester identifies itself to the intermediate gateway and the intermediate gateway identifies itself to the service provider, the target service normally runs with the identity of the intermediate gateway rather than the service requester.

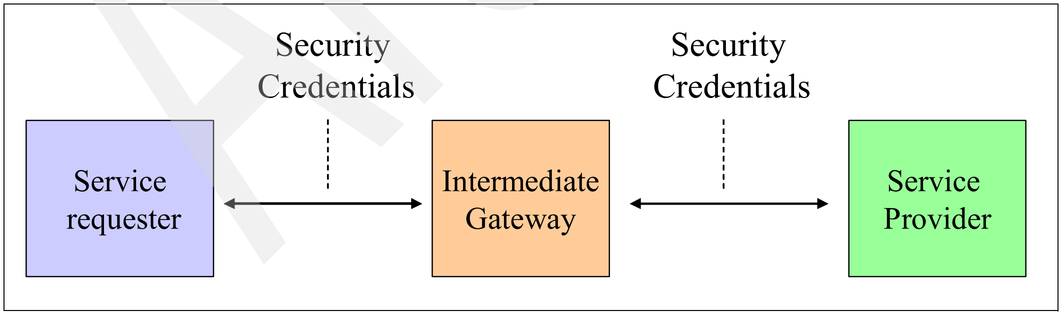


Figure 4-4 Transport-level security with an intermediate gateway

WS-Security addresses this problem by allowing security credentials to be passed within the SOAP message, so that the credentials of the service requester can be passed through an intermediate gateway, and can still be used to identify the requester to the service provider (Figure 4-5).

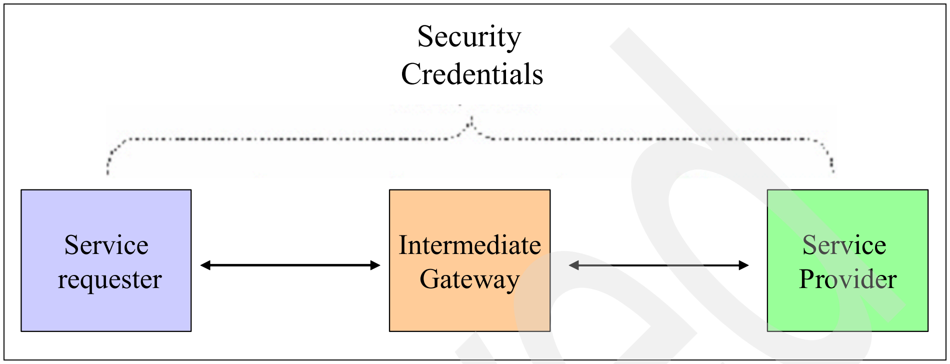


Figure 4-5 SOAP message-level security with an intermediate gateway

Additionally, WS-Security is independent of the transport layer protocol. It can be used for any Web service binding, for example, HTTP, SOAP, Remote Method Invocation (RMI). If you use WS-Security, end-to-end security can be obtained.

Figure 4-6 shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the requester and the service provider. The network portion of the diagram can contain any number of intermediate nodes, some of which might not be trusted.

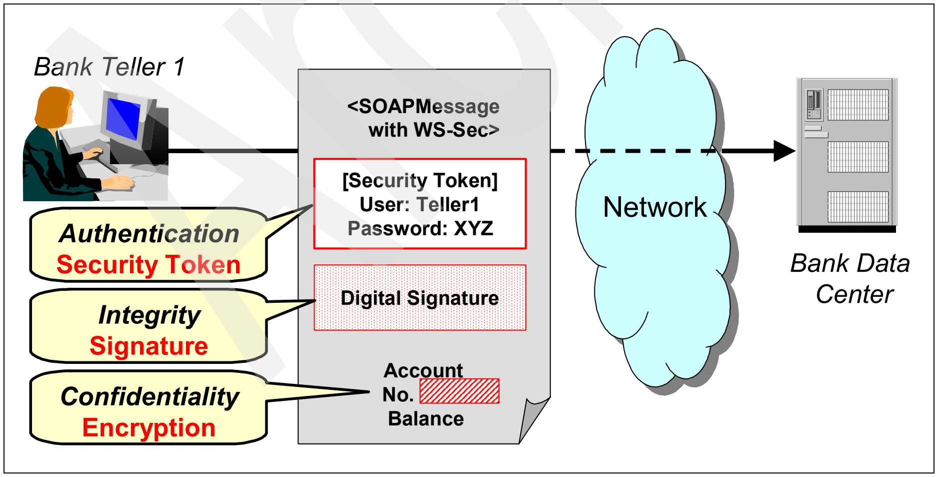


Figure 4-6 An example of a typical scenario with WS-Security

The WS-Security specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), has been proposed by the OASIS WSS Technical Committee. This specification defines a standard set of SOAP extensions. The specification is flexible and is designed to be used as the basis for securing Web services within a wide variety of security models, including public key infrastructure (PKI), Kerberos, and SSL. It provides support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies based on XML signature and XML encryption in order to provide integrity and confidentiality.

The specification includes security token propagation, message integrity, and message confidentiality. However, these mechanisms by themselves do not address all the aspects of a complete security solution. Therefore, WS-Security represents only one of the layers in a complex, secure Web services solution design.

The WS-Security specification defines the use of XML signature and XML encryption:

- Message integrity is provided by XML signature in conjunction with security tokens to ensure that modifications to messages are detected. For more information, refer to:

  http://www.w3c.org/Signature

- Message confidentiality leverages XML encryption in conjunction with security tokens to keep portions of a SOAP message confidential. For more information, refer to:

  http://www.w3c.org/Encryption

For more information about the OASIS consortium, refer to:

http://www.oasis-open.org

4.4.1 Web Services-Security road map

The WS-Security specification addresses only a subset of security services. A more general security model is required to cover other security aspects such as logging and nonrepudiation. The definition of these requirements is explained in a common Web services security model framework, a security White Paper titled *Web Services Security Roadmap*. This road map is described in the following section.

Web Services-Security model framework

The WS-Security model introduces a set of individual interrelated specifications to form a layering approach to security. It includes several aspects of security, including identification, authentication, authorization, integrity, confidentiality, auditing, and nonrepudiation. It is based on the WS-Security specification, codeveloped by IBM, Microsoft, and VeriSign.

The Web services security model is schematically shown in Figure 4-7.

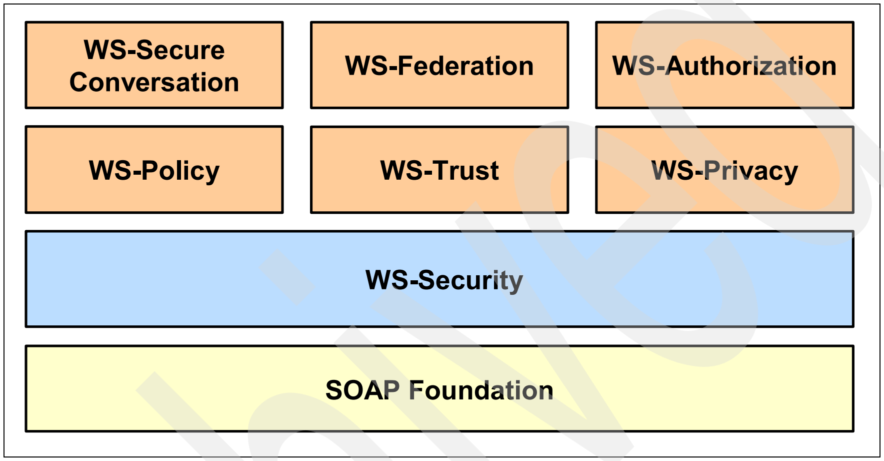


Figure 4-7 WS-Security road map

These specifications include different aspects of WS-Security:

- WS-Policy

  Describes the capabilities and constraints of the security policies on intermediaries and endpoints, for example, the security tokens required, encryption algorithms supported, and privacy rules

- WS-Trust

  Describes a framework for trust models, which enables Web services to securely interoperate, manage trusts, and establish trust relationships

- WS-Privacy

  Describes a model for how Web services and requestors state privacy preferences and organizational privacy practice statements

- WS-Federation

  Describes how to manage and broker the trust relationships in a heterogeneous federated environment, including support for federated identities

- WS-Authorization

  Describes how to manage authorization data and authorization policies

- WS-Secure Conversation

  Describes how to manage and authenticate message exchanges between parties, including security context exchange and establishing and deriving session keys

The combination of these security specifications enables many scenarios that are difficult or impossible to implement with today's more basic security mechanisms such as transport-level security or XML document encryption.

4.4.2 Example of WS-Security

With WS-Security, you can apply authentication, integrity, and confidentiality at the message level. This section provides an example of a SOAP message with WS-Security used for authentication. It also provides a summary of how WS-Security can be used for message integrity and XML encryption.

Authentication

Web services security provides a general purpose mechanism to associate security tokens with messages for single message authentication. A specific type of security token is not required by Web services security, which is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms, for example, a client may provide proof of identity and proof of a particular business certification.

Example 4-1 shows a sample SOAP message without applying WS-Security. The SOAP message is an order request for the CICS-supplied sample catalog application.

Example 4-1 SOAP message without WS-Security

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <p635:DFHOXCMN xmlns:p635="http://www.DFHOXCMN.DFHOXCP5.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

Example 4-1 shows that the SOAP message does not have any SOAP headers. Apply WS-Security by inserting a SOAP security header. WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element <wsse:Security> [1] is the container for security-related information.

When using WS-Security for authentication, a security token is embedded in the SOAP header and is propagated from the message sender to the intended message receiver. On the receiving side, it is the responsibility of the server security handler to authenticate the security token and to set up the caller identity for the request.

Example 4-2 shows the same SOAP message, but this time, with authentication. As you can see, we have a user name and password information contained in the <UsernameToken> element.

Example 4-2 SOAP message with WS-Security

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.
     xsd">
      <wsse:UsernameToken>
        <wsse:Username>WEBUSER</wsse:Username>
        <wsse:Password
         Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1
         .0#PasswordText">REDBOOKS</wsse:Password>
        </wsse:UsernameToken>
      </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <p635:DFHOXCMN xmlns:p635="http://www.DFHOXCMN.DFHOXCP5.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
```

[1] wsse stands for "Web services security extension"

```
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

The <UsernameToken> element of the SOAP message in Example 4-2 contains credentials that can be used to authenticate the user “WEBUSER”. The simplest form of security token is the UsernameToken that is used to provide a user name and password for basic authentication. A WS-Security header processing program can be used to extract a UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header.

A signed security token is one that is cryptographically signed by a specific authority. An X.509 certificate, for example, is a signed security token.

Security token usage for Web services security is defined in separate profiles such as the Username token profile and the X.509 token profile. For further details about these security token standards, refer to:

- Web Services Security: UsernameToken Profile V1.0 (March 2004)

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

- Web Services Security: X.509 Token Profile V1.0 (March 2004)

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

Integrity

Integrity of a message means that the data has not been changed, destroyed, or lost when it is in transit. Essentially, integrity is provided by generating an XML digital signature on a part of the SOAP message. If the message data changes, the signature is no longer valid.

A *digital signature* is a number attached to the message. This signature establishes the following information:

- The integrity of the message

  Is the message intact? That is, has the message been modified between the time it was digitally signed and now?

- The identity of the user who signed the message

  Is the message authentic? That is, was the message actually signed by the user who claims to have signed it?

A digital signature is created through a two-step process:

1. The first step distills a part of the SOAP message, for example, the body, into a large number. This number is the digest code or the fingerprint.

   Several options are available for generating the digest code, for example, the MD5 message digest function and the SHA1 secure hash algorithm. Both these procedures reduce a message to a number.

   The crucial aspect of distilling the document to a number is that if the message changes even in a trivial way, a different digest code results. When the recipient gets a message and verifies the digest code by recomputing it, any changes in the document result in a mismatch between the stated and the computed digest codes.

2. In the second step, the digest code is encrypted with the sender's private key.

This two-step process creates the digital signature, which is appended to the SOAP message before being sent to the service provider.

When the service provider receives the message, it performs the following tasks to verify the signature:

1. Recomputes the digest code for the message
2. Decrypts the signature by using the sender's public key. This decryption yields the original digest code for the message.

   The receiver normally obtains the sender's public key from the sender's X.509 certificate, which is sent as a security token in the SOAP message.

3. Compares the original and recomputed digest codes. If these codes match, the message is both intact and authentic. If not, something has changed and the message is not to be trusted.

Important: The use of digital signatures has a significant impact on the system CPU resource usage.

Confidentiality

Confidentiality of a SOAP message means that it is protected so that only authorized recipients can read the SOAP message. Confidentiality is provided by encrypting the contents of the SOAP message using XML encryption. If the SOAP message is encrypted, only a service that knows the key for confidentiality can decrypt and read the message.

The XML encryption standard specifies a process for encrypting data and representing the result in XML. XML encryption can be used to encrypt any part of a SOAP message, usually, sensitive data such as bank account numbers or user credentials. The result of encrypting data is an XML encryption element that contains or references the cipher data.

XML encryption was published as a W3C recommendation in December 2002. For more information, refer to:

http://www.w3.org/Encryption/2001/

4.5 CICS support for WS-Security

Support for WS-Security is provided by the CICS WS-Security message handler, DFHWSSE1, which is shipped by APAR PK22736. For more information about signature validation and signature generation algorithms, and the decryption and encryption types that CICS supports, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG34-6458.

DFHWSSE1 provides support for digital signing and encryption of the entire SOAP body for outbound messages. It also provides support for the body or elements of the body and header to be encrypted or digitally signed for inbound messages.

Note: CICS TS V3.1 does *not* support Web Services Security for atomic transactions (WS-AT).

WS-Security in CICS can be implemented through the configuration file referenced in the CICS PIPELINE resource definition CONFIGFILE attribute. To do this, add a WS-Security message handler to your pipeline configuration files. This is discussed in detail in 4.5.2, “Pipeline configuration file” on page 146.

4.5.1 Options for securing a SOAP message in CICS

CICS support for WS-Security provides signing and encrypting of SOAP messages. There are several options available. The ones you select depends on the level of security required for the data and the transmission path of the data. CICS-specific support for the following options are described here:

- Basic authentication

  CICS supports service provider mode. The inbound SOAP message header can contain a user name token (*UsernameToken*) consisting of a user name and a password.

  User name tokens are *not* supported for outbound SOAP messages or with CICS as a service requester.

- Signing with X.509 certificates

  CICS supports a service provider mode and a service requester mode. You can provide an X.509 certificate in the SOAP message header to sign the body of the SOAP message for authentication.

- Encrypting

  CICS supports a service provider mode and a service requester mode. You can encrypt the SOAP message body using a symmetric key algorithm such as Triple DES or AES.

  For inbound SOAP messages, an element in the SOAP body can be encrypted first, and later the entire SOAP body. If CICS receives a SOAP message with two levels of encryption, CICS decrypts both the levels automatically. This is not supported for outbound SOAP messages.

  CICS does *not* support inbound SOAP messages that only have an encrypted element in the message header and no encrypted elements in the SOAP body.

- Signing and encrypting

  CICS supports a service provider node and a service requester mode. You can sign and encrypt the SOAP message. CICS signs the SOAP message body first and then encrypts it. This provides message integrity and confidentiality.

4.5.2 Pipeline configuration file

CICS TS uses a pipeline configuration file to handle Web service requests. The configuration file is an XML document and resides in the z/OS UNIX System Services hierarchical file system (HFS). Specify the name of the configuration file in the CONFIGFILE attribute of the PIPELINE definition.

To implement WS-Security in CICS TS, include a <wsse_handler> message handler element and provide configuration information for the handler. To do this, update the configuration file for the appropriate pipeline. DFHWSSE1 uses the configuration information specified for the <wsse_handler> element. The configuration file is made up of different elements. The elements that may be used for WS-Security are:

- <wsse_handler>

  It specifies the parameters used by DFHWSSE1. It can be used in a service provider and service requester pipeline. It contains a <dfhwsse_configuration> element.

- <dfhwsse_configuration>

  It specifies configuration information for DFHWSSE1. It can be used in a service provider and service requester pipeline. It may contain the following optional elements

  - <authentication>

    It specifies the use of security tokens in the headers of inbound and outbound SOAP messages. It can be used in a service provider and a service requester pipeline. In a service provider pipeline, the element specifies whether CICS must use the security tokens in an inbound SOAP message to determine the user ID under which work will be processed. In a service requester pipeline, it specifies that CICS must add an X.509 certificate to the security header for outbound SOAP messages.

    The <authentication> element has two attributes, *trust* and *mode*. These attributes determine whether asserted identity is used and the combination of the security tokens used in a SOAP message. The trust attribute can be set to either none, basic, or signature. The mode attribute can also be set to either none, basic, or signature. For more information about the meaning and valid combinations of these attributes, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

    Asserted identity allows a trusted user to assert or declare that work must run under a different identity (the *asserted identity*), without the trusted user having the credentials associated with that identity. Messages contain a *trust token* and an *identity token*. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run.

> **Note:** If you use asserted identity, the service provider is required to trust the requester to make this assertion. In CICS, the trust relationship is established with security manager surrogate definitions, that is, the requesting identity must have the correct authority to start work on behalf of the asserted identity.

The <authenticate> element can contain the following elements:

  - <certificate_label> (optional)

    This specifies the label associated with an X.509 digital certificate. It is ignored in a service provider pipeline.

  - <suppress/> (optional)

    For the service provider, the handler will *not* use any security tokens in the message to determine under which user ID to run. For the service requester, the handler will *not* add any of the security tokens required for authentication to the SOAP message.

  - <algorithm>

    It specifies the URI of the algorithm that is used to sign the body of the SOAP message.

- <expect_signed_body/>

  This indicates that the <body> of the inbound message must be properly signed. If it is not, CICS rejects the message with a security fault.

- <expect_encrypted_body/>

  It indicates that the <body> of the inbound message must be properly encrypted. If it is not, CICS rejects the message with a security fault.

- <sign_body>

  It directs DFHWSSE1 to sign the body of outbound SOAP messages, and provides information regarding how the messages are to be signed. It can be used in a service provider and a service requester pipeline. It contains the following elements:

  - <algorithm>

    This specifies the URI of the algorithm that is used to sign the body of the SOAP message.

    - <certificate_label>

      This specifies the label associated with an X.509 digital certificate. The digital certificate must contain the private key because this is used to sign the message. The public key associated with the private key is then sent in the SOAP message, which allows the signature to be validated.

  - <encrypt_body>

    This directs DFHWSSE1 to encrypt the body of outbound SOAP messages, and provides information regarding how the messages are to be encrypted. It can be used in a service provider and a service requester pipeline. It contains the following elements:

    - <algorithm>

      This specifies the URI identifying the algorithm that is used to encrypt the body of the SOAP message.

    - <certificate_label>

      This specifies the label associated with an X.509 digital certificate. The digital certificate must contain the public key of the intended recipient of the SOAP message so that it can be decrypted with the private key when the message is received.

Example 4-3, which is taken from *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG34-6458, shows a completed <wsse_handler>, with all the optional elements present. Add this to your configuration file for the pipeline.

Example 4-3 <wsse_handler>

```
<wsse_handler>
 <dfhwsse_configuration version="1">
   <authentication trust="signature" mode="basic">
     <certificate_label>AUTHCERT03</certificate_label>
     <suppress/>
   </authentication>
   <expect_signed_body/>
   <expect_encrypted_body/>
   <sign_body>
     <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
     <certificate_label>SIGCERT01</certificate_label>
   </sign_body>
   <encrypt_body>
```

```
     <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
     <certificate_label>ENCCERT02</certificate_label>
   </encrypt_body>
 </dfhwsse_configuration>
</wsse_handler>
```

The <wsse_handler> element is contained in the <service_handler_list> element. To modify the pipeline configuration file for the CICS-supplied catalog Web sample application, add a <service_handler_list> containing the <wsse_handler>. Example 4-4 shows the original pipeline configuration file basicsoap11provider.xml for the EXPIPE01 service provider pipeline.

Example 4-4 CICS-supplied sample pipeline configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd
">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Example 4-5 shows how to modify the pipeline configuration file to add the <service_handler_list> and <wsse_handler> elements to implement WS-Security. CICS reads the pipeline configuration file and when it finds the <wsse_handler> element, it loads the program DFHWSSE1 from the library SDFHWSLD into your DFHRPL concatenation in order to process the security information. For more information about the elements for the pipeline configuration file, and which ones are contained by other elements (high-level structure diagrams), refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

Example 4-5 WS-Security <wsse_handler> element added to pipeline configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd
">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse_configuration version="1">
          <authentication trust="signature" mode="basic">
            <certificate_label>AUTHCERT03</certificate_label>
            <suppress/>
          </authentication>
          <expect_signed_body/>
          <expect_encrypted_body/>
          <sign_body>
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
              <certificate_label>SIGCERT01</certificate_label>
          </sign_body>
          <encrypt_body>
            <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
              <certificate_label>ENCCERT02</certificate_label>
          </encrypt_body>
        </dfhwsse_configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

If CICS is the service provider, CICS will decrypt any inbound encrypted SOAP message automatically when it processes the message, provided you have the <wsse_handler> element in the pipeline configuration file. The security header in the received message provides all the information required for CICS to decrypt it. In other words, the <encrypt_body> and <sign_body> elements do not have to be specified in the provider pipeline configuration file in order to decrypt the inbound SOAP message. But you can (and probably will want to) include the <encrypt_body> or <sign_body> or both in the provider pipeline configuration file if you want to encrypt or sign or both encrypt and sign the reply body sent back to the requester. This is shown in Example 4-5.

Note: For more information about the pipeline configuration file, details about the elements described earlier, and the message handlers, refer to *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.

4.5.3 Resource Access Control Facility and WS-Security in CICS

Use RACF or your external security manager to create public-private key pairs and X.509 certificates for signing and encrypting outbound SOAP messages, and to authenticate and decrypt signed and encrypted inbound SOAP messages. For more information about this, refer to Chapter 3, “Security technologies” on page 93 of this book, the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458, and *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454.

4.6 Performance considerations

There is a significant extra processing required when you use WS-Security in CICS. Encryption, decryption, and signing of SOAP messages require intensive algorithms that increase the overall response time of your system.

The advantage of using WS-Security is that you can secure a SOAP message through an entire network (end-to-end*)*, which may consist of any number of intermediate nodes. Each node may have a requirement to view the header information of a SOAP message, but you may not want the node to have access to portions of the SOAP body that you protect with WS-Security. Protecting confidential material in the actual SOAP message may avoid the overhead of encrypting and decrypting through SSL at every node (point-to-point*)*.

You may also decide to implement your own security procedures and processing by writing a custom header handler program that can process secure SOAP messages in the pipeline. This method may be a good option if SSL is sufficient to satisfy your integrity and confidentiality requirements. For more information about this, refer to the CICS support Web site by entering the Technote number, 1239021:

http://www.ibm.com/software/htp/cics/support/

Additional information about using a header handler program is also available in *Implementing CICS Web Services*, SG24-7206. This also available on the Web at:

http://www.redbooks.ibm.com/redpieces/abstracts/sg247206.html

An alternative to using WS-Security is to use SSL to encrypt the entire data stream. If you can guarantee that after the data is decrypted, it cannot be accessed by an untrusted node or process, using SSL may be the method that you implement. If, however, you have untrusted nodes in the path or want to hide the details of the data in the SOAP message, use WS-Security.

4.6.1 Optimizing Secure Sockets Layer

Implementing SSL causes an increase in CPU usage. You can optimize the performance of SSL in your environment by performing the following:

- Using only SSL for applications that require encrypted data flows
- Utilizing the IBM eServer zSeries cryptographic hardware that uses the z/OS Integrated Cryptographic Facility (ICSF). You can also customize the encryption settings to use only the cipher suites that use ICSF, such as DES and SHA-1.

  > **Note:** z/OS 1.6 System SSL uses the CP Assist for Cryptographic function (CPACF) directly.

- Increasing the value of the CICS system initialization table parameter SSLDELAY so that the session IDs remain in the cache longer, which results in only partial SSL handshakes.

  > **Note:** The CICS system initialization table parameter SSLDELAY does *not* apply to caching across a sysplex.

- Increasing the value of the CICS system initialization table parameter MAXSSLTCBS so that there are more S8 TCBs in the SSL pool for the SSL handshake negotiation. Refer to 4.6.2, “Performance improvements for Secure Sockets Layer” on page 154 for more information about S8 TCBs.
- Using the CICS SSLCACHE system initialization table parameter to implement SSL caching across a sysplex. If the cache is shared between a number of CICS regions, the throughput of SSL connections improves. Refer to 4.6.2, “Performance improvements for Secure Sockets Layer” on page 154 for more information about SSLCACHE.
- Keeping the socket open by coding SOCKETCLOSE NO on the TCPIPSERVICE definition for the PIPELINE. This is the default for HTTP 1.1 persistent sessions and removes the necessity to perform a full SSL handshake on the second or subsequent HTTP request.
- Only using client authentication by specifying SSL CLIENTAUTH in the TCPIPSERVICE definition when you really want your clients to identify themselves with a client certificate. Client authentication requires more network interchanges during the SSL handshake, and more processing by CICS to handle the received certificate. For more information about increased SSL handshake flows, refer to Figure 4-3 on page 130.

4.6.2 Performance improvements for Secure Sockets Layer

There are performance improvements in CICS for SSL processing, beginning with CICS TS V3.1.

If the client has completed the SSL handshake negotiation with CICS earlier and the session ID is cached, the SSL cache can decrease the time required to establish a connection. If the session ID is still in the cache, only a partial SSL handshake is required. The CICS system initialization table parameter SSLCACHE can be set to either CICS or SYSPLEX. The setting specifies whether SSL is to use the local or sysplex caching of session IDs. Sysplex caching is only useful if multiple CICS socket-owning regions accept SSL connections in the same IP address. The CICS system initialization table parameter SSLDELAY controls how long the session ID remains in the cache.

Note: The CICS system initialization table parameter SSLDELAY does *not* apply to caching across a sysplex.

CICS also supports a greater number of concurrent SSL connections, which can increase the SSL throughput. CICS uses a new open task control block (TCB) mode called SP, which is used for socket thread-owning tasks. The SP TCB manages the SSL pool, owns the Language Environment® (LE) enclave, and owns the SSL cache. SSL processing uses an S8 TCB only for the SSL-specific function. After the SSL negotiation is complete, the S8 TCB is released back into the SSL pool to be reused. The S8 TCBs run as UNIX threads. This allows for many more simultaneous SSL connections than in previous CICS releases. The CICS system initialization table parameter MAXSSLTCBS controls the maximum number of S8 TCBs that can run in the SSL pool.

Note: For more information about the SSLCACHE, SSLDELAY, and MAXSSLTCBS system initialization table parameters, refer to *CICS Transaction Server for z/OS V3.1 CICS System Definition Guide*, SC34-6428.

4.7 Comparison of transport versus message security

This chapter shows that it is possible to implement WS-Security at two levels, the transport-level and the SOAP message-level, with WS-Security. If your Web services environment is simple, for example, it does not span multiple nodes, a security solution based on transport-level security alone may be all that you require. However, for more complex scenarios, this may not be enough by itself.

This section provides general guidelines to help you decide on the type of security solution to implement:

- Use transport-level security only to secure your CICS Web services environment when:
  - No intermediaries are used in the Web services environment
  - The transport is only based on HTTP
  - The Web services client is a stand-alone program

    WS-Security can only be applied to clients that run in a Web services environment supporting the WS-Security specification
- Use WS-Security, probably in addition to transport-level security, when:
  - Intermediaries are used

    Security credentials that flow in the SOAP message can pass through any number of intermediaries. An intermediary can provide an authentication service to CICS, such that the intermediary server authenticates the Web service client and flows an asserted identity to CICS.
  - Multiple transport protocols are used

    WS-Security works across multiple transports and is independent of the underlying transport protocol
  - The Web service partners support WS-Security and a general decision has been taken to flow security tokens in accordance with the WS-Security specification.

5

CICS Web support

This chapter looks at the security issues that you face when you Web-enable your CICS applications using CICS Web support. It also examines the Secure Sockets Layer (SSL) support in CICS Web support and how this integrates with the CICS support of an external security manager (ESM) such as Resource Access Control Facility (RACF). For more information about SSL and RACF security, refer to Chapter 3, “Security technologies” on page 93. For an introduction to CICS Web support, refer to 5.1, “Overview of CICS Web support” on page 159.

This chapter also discusses the way in which CICS Web support addresses the issues of authentication, authorization, data integrity, and privacy. It summarizes the key security considerations you may face, and provides a security matrix and checklist for the issues discussed.

Figure 5-1 shows an overview of the possible Web connections into CICS using CICS Web support.

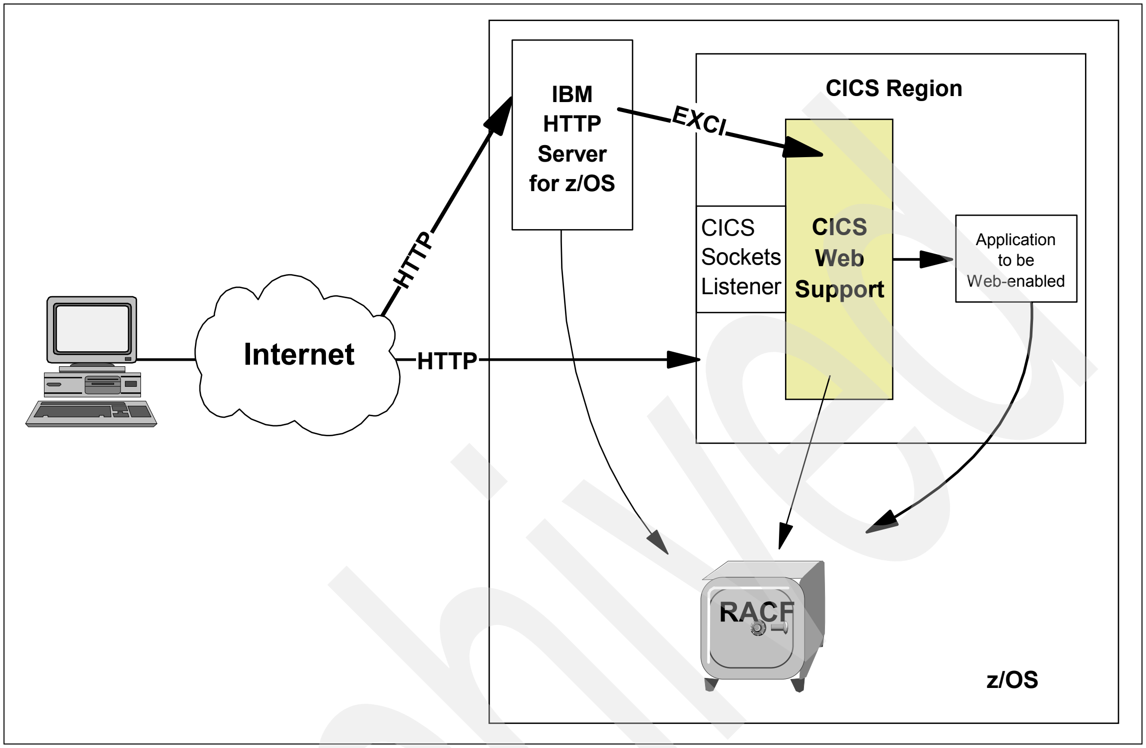


Figure 5-1 Overview of Web connections into CICS using CICS Web support

5.1 Overview of CICS Web support

CICS Web support is a set of resources supplied with CICS Transaction Server (TS) for z/OS. CICS Web support provides CICS with a subset of the Hypertext Transfer Protocol-serving (HTTP-serving) functions found in a general purpose Web server. This allows CICS applications to be invoked by and respond to HTTP requests. A summary of how a CICS application can be Web-enabled using the CICS Web support is illustrated in Figure 5-1.

CICS Web support provides a native HTTP interface to CICS. The interface has its own application programming language (the EXEC CICS WEB API), but can also be used by 3270-based transactions and applications that provide a callable COMMAREA interface. Two different configurations can be used to route the HTTP requests into the CICS region. Both configurations allow the use of the same facilities in CICS, although the configurations of the two options are significantly different. These configurations are:

- A *direct connection* from a Web browser to CICS

  This uses the facilities of the CICS TCP/IP listener to pass the request directly into CICS Web support.

- Through the IBM HTTP Server using the facilities of the *CICS Web Server plug-in*

  This is a CICS-supplied Domino Go Web server API (GWAPI) extension to the IBM HTTP Server. It routes requests from the HTTP Server into the CICS region using the external CICS interface (EXCI). CICS supplies two GWAPI modules called DFHWBAPI and DFHWBDLL. If you use the CICS Web Server plug-in, it is recommended that you use DFHWBDLL because it has support for the FORMS application programming interface (API) commands, for example, EXEC CICS WEB READ FORMFIELD. This book does *not* discuss CICS Web Server plug-in in detail because it is no longer a widely used configuration, and may, in fact, be withdrawn in a future release of CICS.

You can use CICS Web support to invoke three types of CICS applications:

- New applications that use the EXEC CICS WEB programming interface directly

  Such applications may have to have some understanding of the HTTP protocol, but most aspects of the protocol are handled by CICS.

- Existing applications that provide a callable COMMAREA interface

  To use such applications, some new CICS presentation logic must be written. This logic uses CICS facilities to interpret and act on the HTTP request, and then build and return the HTTP response. Application code containing such logic is referred to as *Web-aware*. This Web-aware logic can be contained either within the converter Encode and Decode routines in the original program or in a separate Web presentation module that links to the original program. CICS provides two different methods to create this Web-aware presentation logic:

  - WEB API

    The WEB API, together with the DOCUMENT API and the TCPIP API, provides a rich set of functions to interpret, manipulate, and build the HTTP data streams within a CICS application. These are described in detail in *CICS Transaction Server for z/OS V3.1 CICS Application Programming Guide*, SC34-6433, *CICS Transaction Server for z/OS V3.1 CICS Application Programming Reference*, SC34-6434, and Chapter 6 of *CICS Transaction Server for z/OS V3.1 Internet Guide*, SC34-6450.

  - COMMAREA manipulation

    The COMMAREA manipulation technique was originally introduced with CICS Web Interface (CWI) support in IBM Customer Information Control System/Enterprise Systems Architecture (CICS/ESA®) V4.1. It uses the CICS COMMAREA as a buffer for transferring the HTTP data stream with a range of utility programs to manipulate the data stream. You can use the Hypertext Markup Language (HTML) template manager program, DFHWBTL, to build the response. This technique is still available, but for ease-of-use and higher functionality, it is recommended that you use WEB API.

- Existing applications that were designed to be accessed from 3270 terminals

  To invoke such a *3270 transaction*, the facilities of the 3270 bridge are used. The 3270 transaction remains unchanged, and the 3270 output is converted to HTML. This function is known as the *3270 Web bridge*. This chapter does *not* discuss this any further.

CICS TCP/IP listener

The CICS TCP/IP listener is a part of the CICS Sockets domain and runs as the CSOL (Sockets listener) system task. It provides TCP/IP support to handle requests for internal CICS functions that use TCP/IP services. Currently, this includes HTTP, Internet Inter-ORB Protocol (IIOP), and external call interface (ECI) over TCP/IP. As such, it is not a component of CICS Web support, but a service used by CICS Web support.

The CICS TCP/IP listener can listen on many different ports simultaneously. The attributes of the expected traffic on each port are specified by a TCPIPSERVICE resource definition.

> **Note:** The CICS TCP/IP listener is completely separate from, and must *not* be confused with the TCP/IP Socket Interface for CICS, which provides an application-level TCP/IP socket interface for CICS applications, and is described in detail in *z/OS V1R7.0 Communications Server: IP CICS Sockets Guide*, SC31-8807.

URIMAP resource definitions

URIMAPs are resource definitions that were introduced in CICS TS V3.1. They are used to simplify the analysis of incoming URLs[1] and their mapping to CICS resources. By using a URIMAP, you can map a URL to one of the following:

- A dynamic response that is produced by running an *alias* transaction
- A dynamic response that is produced by scheduling a sequence of programs specified in a PIPELINE definition. (This is the basis for the CICS Web Services support discussed in Chapter 4, “CICS Web services” on page 121.)
- A static response that is created from a CICS document template (DOCTEMPLATE) or a file in the z/OS UNIX system services hierarchical file system (HFS)
- A redirection to another URL, possibly on a completely different server

If the URIMAP selects a dynamic response, it may also specify the alias transaction name, a converter program name, and an application program name. It can also select the user ID under which the alias transaction must run. These are the attributes of the alias transaction that had to be selected by the Analyzer program in CICS TS V2.

URIMAPs can be managed by CEMT or CICSPlex® System Manager (CICSPlex SM), which allows for an easy way to enable or disable access to certain URLs and the associated applications. A URIMAP can also be temporarily changed to a redirection type that you can use to offload the URL to another location.

Web attach transaction (CWXN)

The Web attach transaction performs Web attach processing. Its transaction name is normally CWXN, but it can have an alternate name specified in a TCPIPSERVICE definition. It processes the incoming URL and matches it against the installed URIMAP definitions. CWXN invokes the DFHCCNV code page conversion routine to convert HTTP headers. It writes the data received from the client to temporary storage queues for later use by the WEB API

[1] Note that a URL is a special case of a URI. A URL always corresponds to a physical resource that is located on a server. A URI may also identify an abstract entity such as an encryption algorithm.

commands, and then searches the URIMAPs to find a match. If a matching URIMAP is found and it specifies a static response or a redirection, the response is processed immediately in the CWXN transaction. Because there is no necessity for an alias task in this case, it is not attached.

If there is no match or if the matching URIMAP requests it, the Analyzer program is called. DFHCCNV is invoked again to convert the user data, and the alias task is attached. If a static response is returned and the connection is persistent, then, CWXN issues an asynchronous receive to wait for more requests from the client. Irrespective of whether the alias task is attached or not, CWXN terminates. It is reattached if more data is received on the socket.

DFHCCNV

The DFHCCNV code page conversion routines are invoked by the Web attach processing to convert the American Standard Code for Information Interchange (ASCII) HTTP headers and user data of the Web browser client to EBCDIC, and by the alias transaction to convert EBCDIC output back to ASCII. This conversion only affects the data in the COMMAREA. It does not affect the data in the temporary storage queues that are used by the WEB API commands.

Analyzer

The purpose of the Analyzer is to analyze the incoming HTTP requests. It runs under the CWXN transaction, and decides whether the requests will be accepted, and if they will be, the parameters that will be set. Among other things, it can set the name of the alias, converter, user ID, and application program. If URIMAPs are being used, it is possible that the Analyzer is not called at all.

Alias task

The alias task is attached by the CWXN task if it did not process the HTTP request by itself. The default alias transaction code is CWBA. However, URIMAP or Analyzer Processing can modify this. The alias initially invokes the program DFHWBA, which links to the business logic interface (BLI). The BLI is an externally callable interface to CICS Web support, implemented by the module DFHWBBLI. It provides a mechanism for implementing Web-aware presentation logic in the *converter* program. The converter provides Decode and Encode routines to receive the HTTP request and construct a COMMAREA for the target application and to take the COMMAREA from the application, and send out an HTTP response.

Note: It is possible to bypass the converter and implement the Web-aware logic in a separate module that communicates directly with the business logic through a COMMAREA interface.

5.2 CICS Web support security issues

This section looks at the two main security issues that you face when accessing your CICS applications using CICS Web Support, that is, using SSL support to provide encryption of the data being sent and received, and authenticating and authorizing the user. Although it is possible to access the facilities of CICS Web support from the IBM HTTP Server for z/OS by using a CICS-provided plug-in, this is no longer a recommended technique. In fact, this chapter discusses only the use of direct HTTP connections into CICS, although the CICS Web support security decision matrix in Table 5-1 on page 178 compares them both.

Figure 5-22 shows security with CICS Web support.

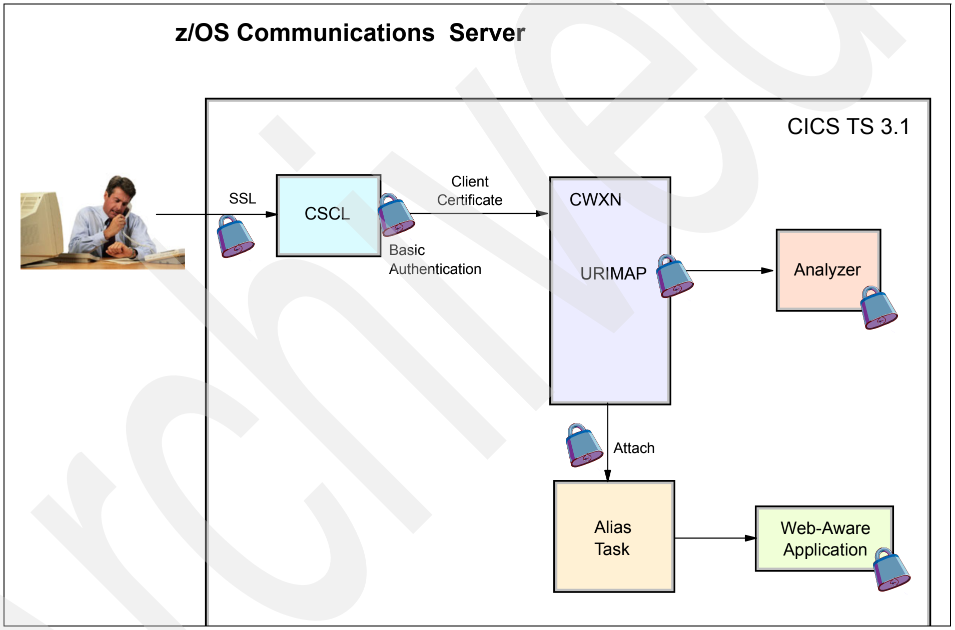


Figure 5-2 Security with CICS Web support

5.2.1 Secure Sockets Layer support

SSL, and its more modern equivalent, Transport Layer Security (TLS), are described in 2.13, “Transport Layer Security 1.0 protocol” on page 70. For the CICS server to use SSL, you must first create a server certificate. To control the strength of encryption that is to be used in the SSL connection, you must also specify an appropriate CIPHERS list in the TCPIPSERVICE definition, as described in 2.13.2, “Cipher suites” on page 72.

> **Important:** Specify TCPIP=YES and the KEYRING system initialization parameters for SSL to be available in your CICS region.

To activate SSL support for an *incoming* HTTP request, specify one of the following as the value of the SSL parameter of the TCPIPSERVICE definition:

- YES

  If you set the value to `YES`, CICS sends a server certificate to the client.
- CLIENTAUTH

  If you set the value to `CLIENTAUTH`, CICS sends a server certificate to the client and also requests the client to send a certificate to CICS. You can affect the way CICS handles the client's certificate by using the AUTHENTICATE parameter.

If you are using the *outbound* HTTP support, and the remote server requests a client certificate, specify the certificate label in the CERTIFICATE parameter of EXEC CICS WEB OPEN or on a URIMAP that is referenced by EXEC CICS WEB OPEN. If you do not specify it in either place, the default certificate in the key ring, if any, will be used.

5.2.2 Identifying and authenticating the client user

When using a direct connection to CICS Web support, you can determine the user ID the alias task runs under in one of the following ways:

- The user ID and password are requested using HTTP basic authentication
- The user ID is associated with an SSL certificate supplied by the client
- The user ID is specified in the URIMAP definition
- The user ID is selected by the Analyzer program. The Analyzer runs only if no matching URIMAP is found or if the matching URIMAP specifies ANALZER(YES). If the Analyzer is executed, it receives the user ID selected by one of the earlier methods as an input. The Analyzer may either choose to leave that user ID unchanged or may select a new one.

The user ID selected for the alias transaction is used when executing the converter program and also when executing your business application program.

Figure 5-3 shows the authentication options available in CICS Web support.

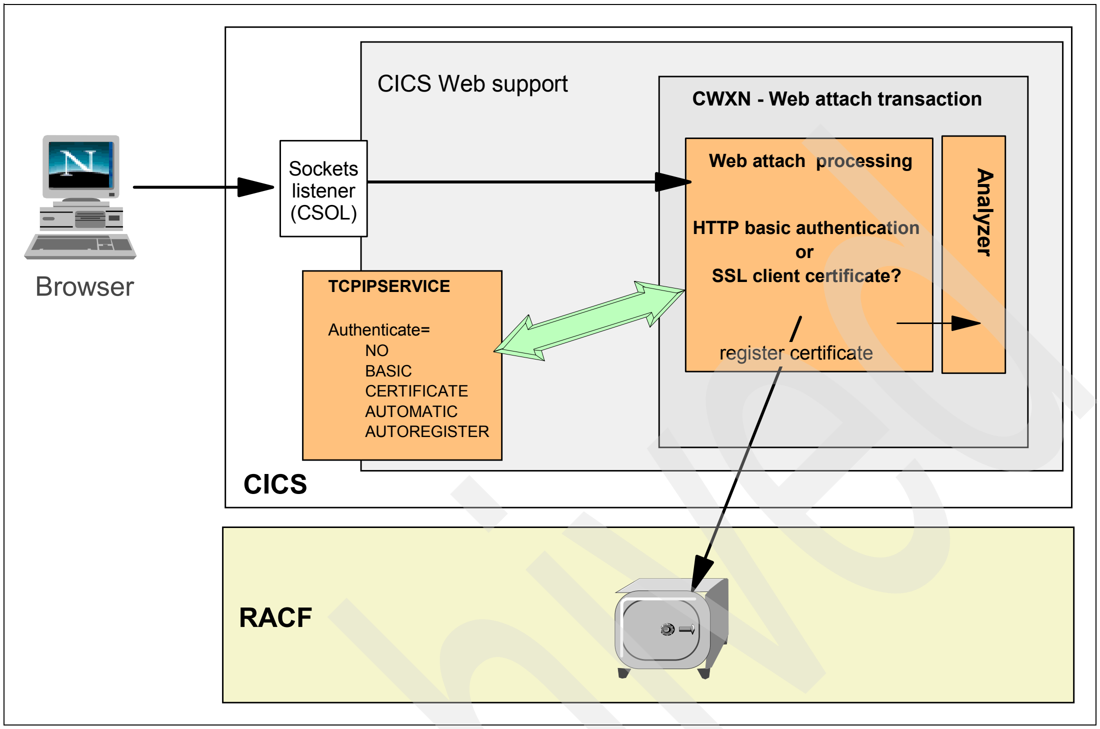


Figure 5-3 Authentication options for CICS Web support

Using HTTP basic authentication

Basic authentication is an HTTP feature in which the user ID and password are transmitted over the network in a scrambled format that uses the Base64 encoding scheme. It is, however, easily unscrambled. If you want to use HTTP basic authentication, specify `BASIC` as the value for the AUTHENTICATE parameter of the TCPIPSERVICE definition.

You can specify basic authentication irrespective of whether you use SSL or not, but because the password is so easily deduced, it is really only safe to use basic authentication in combination with SSL.

Mixed case passwords

If you are using z/OS 1.7 or later, RACF includes optional support for mixed-case passwords. When this option is activated, CICS does not fold passwords to uppercase, but passes them to RACF in the same way they are entered. This change affects the passwords you enter in a basic authentication dialog. However, it does not affect the processing of user IDs, which continue to be folded to uppercase.

Using a Secure Sockets Layer client certificate

Using SSL with CICS Web support enables the client to supply a digital certificate in order to identify itself. If you want to use client certificates, specify `SSL(CLIENTAUTH)` for the TCPIPSERVICE definition used to define the relevant CICS TCP/IP listener. Also specify one of the following values for the AUTHENTICATE parameter:

– `CERTIFICATE`

  The SSL client certificate is used to authenticate and identify the client. The client must send a valid certificate that is already registered to the security manager and associated with a user ID. If a valid certificate is not received or the certificate is not associated with a user ID, the connection is rejected.

  When the user is successfully authenticated, the user ID associated with the certificate identifies the client.

– `AUTOREGISTER`

  The SSL client certificate is used to authenticate the client.

  - If the client sends a valid certificate that is already registered to the security manager and associated with a user ID, that user ID identifies the client.
  - If the client sends a valid certificate that is not registered to the security manager, HTTP basic authentication is used to obtain a user ID and password from the client. If the password is valid, CICS registers the certificate with the security manager and associates it with the user ID. The user ID then identifies the client.

– `AUTOMATIC`

  This combines the AUTOREGISTER and BASIC functions.

  - If the client sends a certificate that is already registered to the security manager and associated with a user ID, that user ID identifies the client.
  - If the client sends a certificate that is not registered to the security manager, HTTP basic authentication is used to obtain a user ID and

password from the client. If the password is valid, CICS registers the certificate with the security manager and associates it with the user ID. The user ID then identifies the client.

- If the client does not send a certificate, HTTP basic authentication is used to obtain a user ID and password from the user. When the user is successfully authenticated, the user ID that is supplied identifies the client.

Using a URIMAP

The USERID attribute of the URIMAP resource specifies the user ID of the attached alias task. This user ID applies to all the inbound requests that match the SCHEME, HOST, and PATH specified in the URIMAP. This is only used when the specific user does not have to be authenticated, but a user ID other than the CICS default user ID is required to authorize access to the associated resources.

An example of when this can be used is if the CICS default user ID is not authorized to run any Web alias transactions. If the real user is to be authenticated using an HTML forms-based dialog, an alias transaction is required. A special user ID can be set up to allow a specific alias transaction and associated programs to be run before the real user ID is established.

Using the Analyzer

For special converter programs or a matching URIMAP definition that specifies that the Analyzer must not be called, the Analyzer is bypassed. If the Analyzer is called, it can use any information in the incoming HTTP request or can obtain it using the EXEC CICS WEB and EXEC CICS TCPIP commands to determine the user ID that must be used for the alias task.

The Analyzer can also determine whether the user must supply the user ID and the password. This is done through HTTP basic authentication or a HTML forms-based dialog. CICS ships sample Analyzer programs for both alternatives, but you may want to write your own Analyzer. An example is provided in the CICS SupportPac™ CA8D, which is available at:

http://www.ibm.com/software/ts/cics/txppacs

Note: You must write or customize an Analyzer program to authenticate the user only if the other methods of authentication are unsuitable. The Analyzer program can perform other functions though, and may be required even if they are not used for authentication, for example, it is a good place to produce an audit of Web access to your CICS region.

5.2.3 Customizing basic authentication prompts

Standard HTTP basic authentication does *not* contain a protocol for expired passwords. However, if you use a z/OS security manager such as RACF, your user passwords are likely to expire on a regular basis. CICS detects expired passwords during basic authentication and enters into a dialog with the user who is outside the normal protocol.

When CICS detects an expired password, it changes the converter program to DFHWBPW. This program runs instead of the expected application program, and returns an HTML form in which the user can enter new password information. The page that is returned is constructed from a number of document templates, and it is recommended that you customize these templates to meet your company's requirements.

It is important to know how a browser usually handles basic authentication. When an HTTP 401 response is received from a server, the browser pops up a small window in which the user must enter the expected user name and the password. The browser saves the values the user enters, and uses them in all the subsequent requests that are sent to that server for the entire lifetime of the browser program. The saved user names and passwords are kept in an *authenticated session cache*. Because password expiry is not a part of the standard basic authentication protocol, entities in the authenticated session cache are not usually changed until the browser session is restarted. When CICS changes an expired password, not only does it have to change the password in the security manager, it also has to arrange for the password to be updated in the browser's authenticated session cache.

Processing is performed as follows:

1. The browser sends CICS a user name and a password from its authenticated session cache, using the normal basic authentication header.
2. CICS detects that the password is expired, and changes the converter program to DFHWBPW.
3. DFHWBPW sends back a form in which the user must enter the user name, the expired password, and two copies of the new password.
4. The user enters the required details, and CICS executes an EXEC CICS CHANGE PASSWORD command to change the expired password.
5. At this point, the password is changed in the security manager, but not in the browser's authenticated session cache. Note that CICS has not yet returned an HTTP 401 status code to indicate that the password is invalid. Therefore, CICS sends back a message to warn the user about what is going to happen next.

CICS returns a message warning that the user will shortly be prompted again for a password, and that the *new* password must be entered. This message contains a button that reissues the original request that caused the expired password to be detected in the first place.

6. The user must respond to the warning message by pressing the button. This causes the original request to be submitted again.
7. The browser does not know that the password has changed. It therefore, sends the old expired password with the reissued request. This time, CICS does not detect that the password is expired because the password is now changed. Instead, it detects it as an invalid password, and issues an HTTP 401 response.
8. The browser detects the 401 response and throws up a new prompt for the user name and the password. This is when the user must re-enter the new password.
9. The browser updates its authenticated session cache with the new password. All the subsequent requests from the browser use the new password.

Because CICS uses document templates to construct its messages, you can customize the messages by changing the document templates. Following are the templates that CICS uses:

- DFHWBPW1

  Contains an HTML prolog for the expired password prompt page, and initial values for all the messages that may be issued

- DFHWBPW2

  Contains an HTML form in which the user must enter the user name and the old and the new passwords

- DFHWBPW3

  Contains the warning that the password has been updated in the server and that the user will be prompted for the new password. This template is used when the original request has a method of GET and contains a refresh tag that automatically redirects the user back to the original request after a short interval.

- DFHWBPW4

  Contains the warning that the password has been updated in the server and that the user will be prompted for the new password. This template is used when the original request has a method of POST, and replicates the data entered in the original form. It does not contain a refresh tag, but does contain a button that the user must click to continue with the original request.

The initial prompt for the user to replace the expired password is composed from template DFHWBPW1, followed by a symbolic reference to one of the messages initialized by DFHWBPW1, followed by template DFHWBPW2.

You must change DFHWBPW1 if you want to change the style of the password expiry page to match your corporate standards, or if you want to customize the messages issued by CICS when it detects errors in the supplied password. You can, for example, change the messages to your language. You probably do not have to change template DFHWBPW2. However, you must change DFHWBPW3 and DFHWBPW4 if you have the necessity to change the style or content of the pages that warn the user about having to re-enter the password in the browser after it is changed in the server.

5.2.4 Authorization with a direct connection

Figure 5-4 illustrates the points within a CICS Web support direct connection architecture, where it is possible to perform authorization checks. In the following discussion, it is assumed that a user ID has been assigned to the client who has sent the HTTP request to CICS, using one of the methods described in 5.2.2, “Identifying and authenticating the client user” on page 164.

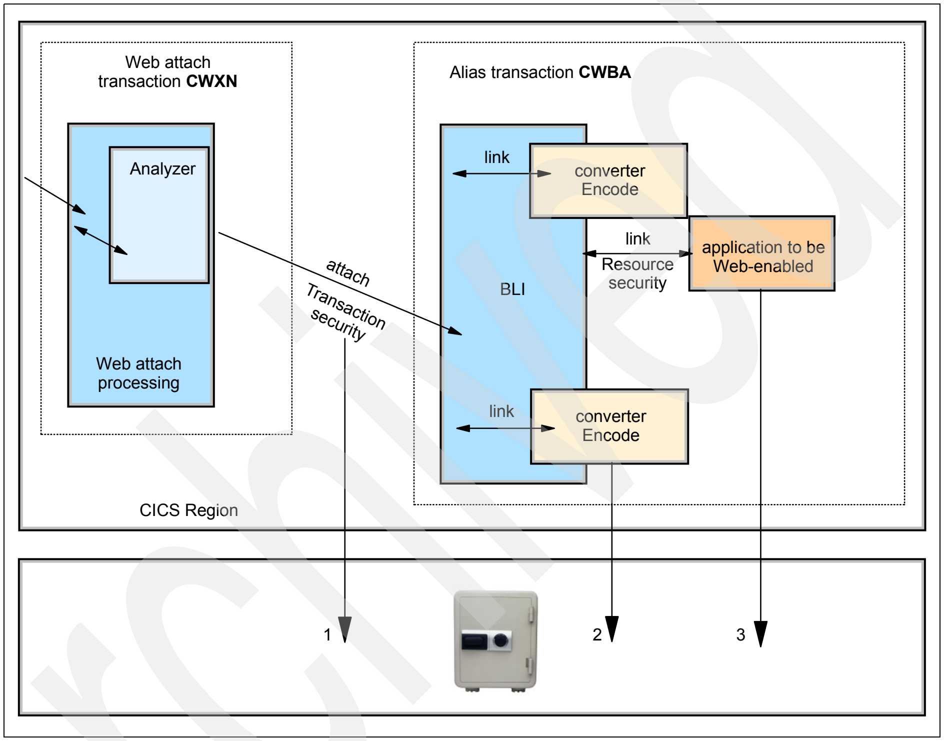


Figure 5-4 Authorization with a CICS Web support direct connection

Following are the authorization checks you can perform:

- Attaching the alias transaction

  It is possible to protect the alias transaction using transaction security. The client user ID must be authorized to attach the alias transaction. To provide flexibility in your security definitions, use the URIMAP or Analyzer to assign different alias transaction names for different URLs. You then have control over which users are permitted to access which URLs by controlling their access to the associated alias transaction. All the alias transactions that you specify for this purpose must have an initial program of DFHWBA.

- Link-to-user program

  The programs to which the BLI is permitted to link can also be protected by using resource security for programs. This permits a finer level of authorization control than transaction security. For resource security to be effective for the alias transaction, change the RESSEC attribute of the alias transaction to RESSEC(YES). Because you cannot modify the IBM-supplied definition of CWBA in the DFHWEB RDO group, in order to change its RESSEC attribute, you must first copy it to a user-defined group.

- Application security

  Within the application, use the resource security to control access to resources such as files, queues, and LINKed-to programs. You can use command security to prevent unauthorized access to CICS administrative functions. In addition, you may require application-specific security such as user data or user-defined classes in RACF, or by relating something in the incoming data to be checked against the data held in the application database.

The authorization of CICS Web support requests differ slightly in emphasis, depending on whether you are using a Web-aware program or the 3270 Web bridge to run the existing 3270-based transactions. These differences are discussed in the subsequent sections.

Web-aware applications

Web-aware applications can understand HTTP without using the 3270 Web bridge. They can use the CICS WEB API, DOCUMENT API, or TCPIP API calls to process the data entering CICS, or they can accept data into CICS as a COMMAREA containing HTTP requests.

The primary defense against unauthorized access is still the protection of the alias transaction, which is coded in the Analyzer. It is recommended that there be at least two, one used when the application server program accesses confidential data, and another when the data being accessed is public. This allows transaction-level security to provide you with the first line of protection.

The next level of security that you can use is to protect the user program LINKed-to from the BLI. Because this is the wbra_server_program set in the Analyzer, it is already protected if the Analyzer is coded to restrict the programs that are accessed.

As you descend within the user program, you can continue to use CICS-provided security. Depending on the design of the application, it may be necessary to provide resource-level protection for the resources accessed or the programs LINKed to within the application. Additionally, use the TCP/IP API and the WEB API to obtain information about the Web user, for example, the client IP address, the port in use, the SSL session usage, and the SSL certificate information. The information that is obtained may be used with normal CICS application interfaces to RACF in order to obtain more details about the user and to make application-based authorization decisions. It can also be compared to application-specific information, for example, client number or perhaps a field from a cookie, in order to determine which application-specific information this client is allowed to access.

Web-owning region

A good design that you can utilize when using Web-aware programs is a *Web-owning region* in order to authenticate all the Web requests and then route all the requests to a remote region using a distributed program link (DPL) request. This provides a physical separation between the Web world and the existing applications, and allows authentication to be performed in the Web-owning region and authorization to be performed in the application-owning region (AOR), the same way that a terminal-owning region (TOR) can be used to authenticate users in a 3270 network.

In addition, this design enables you to utilize link security, whereby a link user ID defined in the AOR connection can provide an additional limit to the maximum authorization possible from the connected Web-owning region.

If you are using a Web-owning region architecture, you must be aware that all the Web-aware applications and all the transactions running under the 3270 Web bridge facility must run in the WOR, and only the distributed program link (DPL) calls to the business logic or function shipping requests can be sent to the AOR. Figure 5-5 illustrates a sample Web-owning region-application-owning region architecture.

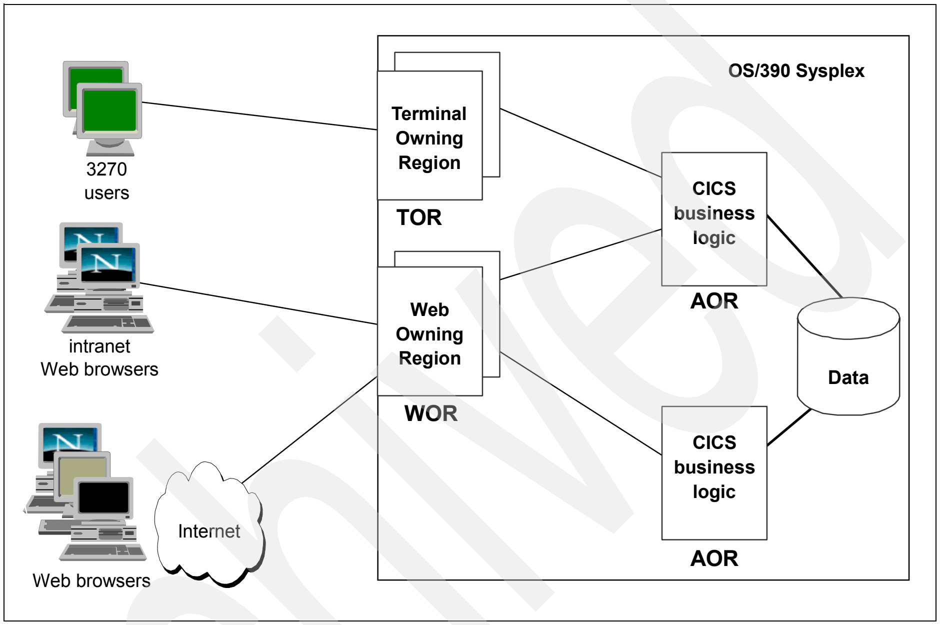


Figure 5-5 A Web-owning region design

3270-based applications

The 3270 Web bridge allows 3270-based transactions to run as Web transactions with little or no modification. Instead of a 3270 terminal, a buffer known as the bridge facility is used, which appears to be a 3270 screen to the application. The user ID associated with this bridge facility is taken from the Analyzer wbra_userid field and stored in the sign-on extension of the bridge facility. Thus, the bridge facility resembles a 3270 terminal with preset security.

Because the bridge facility behaves like a terminal with preset security, it is not possible to run a program that issues the EXEC CICS SIGNON command, or to use the CESN transaction. Thus, any existing 3270-based transactions you use through the 3270 Web bridge may require modification so as to not use the explicit EXEC CICS SIGNON logic. Figure 5-6 illustrates the points within a CICS

Web support direct connection 3270 Web bridge application, where it is possible to perform authorization checks.

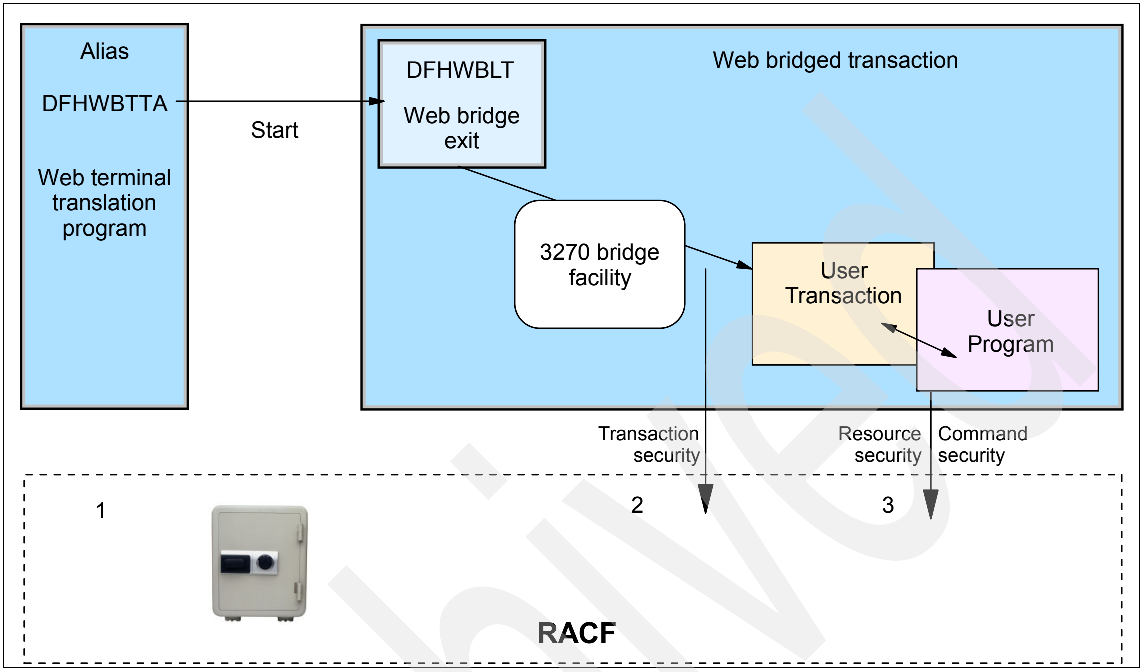


Figure 5-6 CICS Web support 3270 Web bridge authorization checks

The following list refers to the numbered points in Figure 5-6:

- CICS resource security ("1") protects the EXEC CICS LINK from the BLI to DFHWBTTA because this is equivalent to the user program for a Web-aware application.
- CICS transaction security ("2") protects the 3270 transaction started under the bridge. Because the transaction and the bridge exit are running under the user ID set by the Analyzer, this does not appear different from the standard 3270 access in terms of security.
- Resource or command security ("3") protects the resources within the program that the 3270 transaction runs. The major difference here is that 3270 transactions are not Web-aware, and so do *not* use the TCP/IP and WEB APIs. Also, they have no concept of the user being a Web user. Normal CICS application interfaces to RACF, for example, EXEC CICS QUERY SECURITY, can still be used to obtain more details about the user and make application-based authorization decisions.

Note that because HTTP is an inherently stateless protocol, multiple screens within a pseudo-conversational transaction must be linked together by state data, which is managed by the 3270 Web bridge. This state data is managed by the Web state manager program DFHWBST, which holds this information in CICS temporary storage. This state data is timed out at the interval set by the System Initialization Table (SIT) parameter WEBDELAY. Thus, if a pseudo-conversational transaction is left suspended for longer than this period, it is terminated and the user receives an HTTP 500 response. This offers a degree of protection for 3270 Web-enabled transaction by providing a facility to time-out inactive users.

Some CICS applications make wide use of the 3270 terminal ID for controlling the application. Often, this can be preset, based on a pool of Systems Network Architecture logical units (SNA LUs) or hard-coded for a particular terminal. However, when using the 3270 Web bridge, the terminals are autoinstalled and the terminal IDs follow the form “}AAA” to “}999”, although this is not documented and may be subject to modification. Thus, any application decisions based on the terminal ID must be carefully considered if you are Web-enabling them by using the 3270 Web bridge.

5.2.5 Access to static content

If you use a URIMAP to specify a static response from a DOCTEMPLATE or an hierarchical file system (HFS) file, note that access to the static data by the client user ID cannot be controlled in CICS TS V3.1. Resource security does *not* apply to DOCTEMPLATEs, and the access control on HFS files applies only at the address space level, that is, the HFS permissions are only applied to the CICS region user ID.

You must therefore, be especially careful when using a generic path. When matching to externalize a HFS directory structure, make sure that all the files in the directory are intended to be publicly visible if they are accessible by the CICS region user ID.

5.2.6 Design issues

In summary, following are the key points to be considered when designing a secure solution with a CICS Web support direct connection:

- A DMZ is a key part of a secure Web-enablement strategy. With CICS Web support, you can participate in this architecture by installing a dedicated CICS region, the *listener region,* in a separate Web logical partition (LPAR). As is the case with a TOR, the listener region not only serves to handle incoming requests from the Web, it also acts as a protocol switch, converting TCP/IP requests into internal CICS protocols.

After the listener region has authenticated the user ID, a CICS user program can be invoked in another CICS region using a distributed program link (DPL) call, as shown in Figure 5-7. This program can be on a different LPAR within the same sysplex if the DPL call utilizes the cross-system coupling facility (XCF) communications.

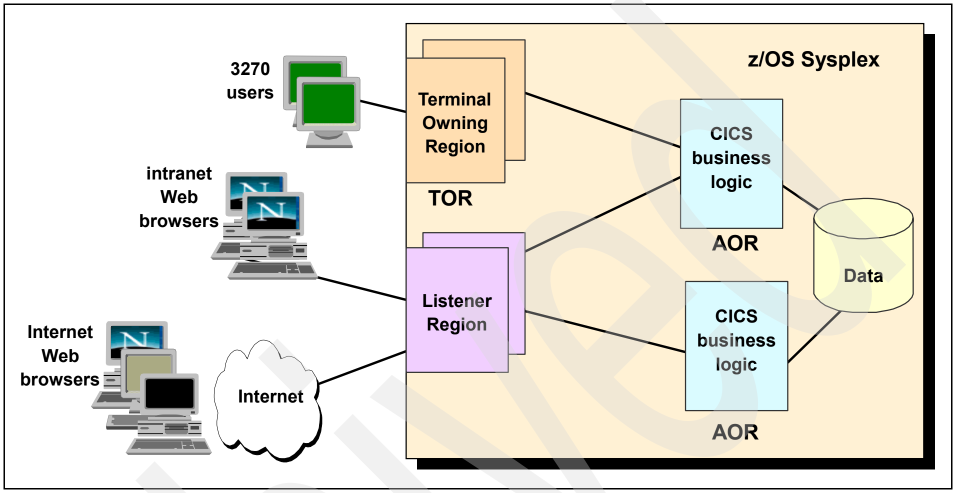


Figure 5-7 Sample CICS Web support listener region configuration

> **Note:** A separate listener region is possible only if the business logic is a COMMAREA-based application. Applications using the EXEC CICS WEB API cannot be run remotely.

- Another argument for setting up a listener region may be to have link security between the listener region and the AOR where the business logic runs. The link user ID carries the maximum level of authorization for any transaction in the AOR. All the authorizations in the AOR are checked against the link user ID and the user ID flowed with the DPL request. Both authorizations are required for the business logic to be called.
- Also consider using URIMAPs, or, if you are just using the default Analyzer, disabling the program autoinstall. This is because the URL format expected by the default Analyzer allows any CICS program to be invoked. Obviously, you are unlikely to want just any Web browser to invoke any program that can be found in your CICS load libraries.

5.3 Designing a secure solution

This section provides a decision matrix to summarize the security support of CICS Web support. It then provides a checklist of implementation tips that you must consider when Web-enabling CICS applications using CICS Web support.

5.3.1 CICS Web support security matrix

The decision matrix shown in Table 5-1 summarizes the key decision points for securing Web access to CICS using either a CICS Web support direct connection or the CICS WebServer plug-in. Each item is discussed further subsequently.

Table 5-1 CICS Web support security decision matrix

Architectural questions	**CICS Web support direct connection**	**CICS Web server plug-in**
Authentication of client by user ID and password	Yes, using AUTHENTICATE(BASIC) or custom Analyzer	Yes, through IBM HTTP Server support for basic authentication or custom HTML form application
SSL encryption	Yes, both CICS and the IBM HTTP Server support SSL encryption, and both can utilize IBM System z cryptographic hardware	
SSL client certificate support	Yes, using AUTHENTICATE(AUTOREGISTER) or AUTHENTICATE(CERTIFICATE) and manual registration	Yes, using HTTP Server SSL support
Flowing user ID into CICS	Not applicable because user authenticated within CICS	Yes, flow security context from HTTP Server to CICS using ATTACHSEC=IDENTIFY
Handling unregistered SSL client certificates	Yes, with SSL(CLIENTAUTH) and AUTHENTICATE(NONE)	Indirectly, with the RACF certificate name filtering feature
Mapping of many Web users to one RACF user ID	Yes, with AUTHENTICATE(CERTIFICATE) and RACF certificate name filtering	Indirectly, with the RACF certificate name filtering feature
Restricting maximum Web user authorization	Yes, using a customized Analyzer or for Web-aware applications, by using a Web-owning region and a link user ID	Yes, using either a surrogate user in the HTTP Server *Protect* directive, or a link user on external CICS interface (EXCI) connection
Handling expired password	Yes, built-in feature that can be customized in document templates DFHWBPW1-4	Use the HTTP Server pwapi.so sample plug-in or HTTP basic authentication

Architectural questions	CICS Web support direct connection	CICS Web server plug-in
Application access to authentication data	Yes, for Web-aware applications only, using the CICS WEB and TCPIP APIs.	No, although HTTP Server protection directives can be used
Signon support in CICS	VERIFY PASSWORD only. Any existing CICS SIGNON logic must be modified.	SIGNON logic must be modified to use flowed HTTP Server authentication
Denial of service by uploading large HTTP request files	Use the TCPIP APIs	

Authentication of client by user ID and password

When using a CICS Web support direct connection, you can use the built-in basic authentication support by specifying `AUTHENTICATE(BASIC)` in the TCPIPSERVICE definition, or write your own Analyzer to implement basic authentication or HTML forms to prompt for a user ID and password. When using the CICS Web Server plug-in, you can use the HTTP Server support for HTTP basic authentication or write your own custom HTML form application.

Secure Sockets Layer encryption

CICS Web support supports SSL encryption with both a direct connection and the CICS Web Server plug-in. Both the methods can utilize the System z cryptographic hardware in order to improve performance substantially when using SSL.

Secure Sockets Layer client certificate support

CICS Web support supports authentication using SSL client certificates with both a direct connection and the CIS WebServer plug-in.

Flowing user ID into CICS

When using a CICS Web support direct connection, this is not necessary because authentication is performed within CICS and is passed to the invoked CICS Web support programs using the wbra_userid field. When using the CICS Web Server plug-in, authentication is performed within the IBM HTTP Server. The authenticated user ID must then be flowed into the CICS region by specifying `ATTACHSEC=IDENTIFY` on the EXCI connection definition.

Handling unregistered Secure Sockets Layer client certificates

This is supported when you use a CICS Web support direct connection and the CICS Web Server plug-in. If you are using a CICS Web support direct connection, use the CWS Certificate Autoregistration feature, whereby the Web user's client certificate can be automatically registered to a known user ID. Alternatively, you can write your own customized Analyzer program to perform your decisions about unregistered certificates. With the IBM HTTP Server or a CICS Web support direct connection, use the RACF certificate name filtering feature. This automatically assigns a RACF user ID to a Web user according to the predefined rules relating to the information contained in the certificate.

Mapping of many Web users to one Resource Access Control Facility user ID

To achieve this, use the SSL client certificates and the RACF certificate name filtering feature. In addition, when you use a CICS Web support direct connection and the CWS Certificate Autoregistration feature, many Web users can be mapped to one RACF user if each Web user knows the required password.

Restricting the maximum permissions of a Web user

You may want to prevent a user from running a transaction from the Web for which that user ID is authorized to run in a non-Web environment. With the CICS WebServer Plugin, you can achieve this by using a CICS Link user ID in the SESSIONS definition, and granting a maximum authority to this user ID. However, note that the Link user ID is discarded and not checked, when a transaction is run under the 3270 Web bridge.

With a CICS Web support direct connection, the same can be achieved using a Web-owning region and an MRO connection to the AOR. Alternatively, you can limit the maximum authority of a Web user by using a customized Analyzer program, which can check if the requested function must be available to a Web user before authenticating the request.

Handling expired passwords

A CICS Web support direct connection handles expired passwords automatically. When it detects an expired password, it prompts the user to re-enter it without using HTTP basic authentication. You can customize this process. See 5.2.3, "Customizing basic authentication prompts" on page 168 for more details. If you are using the CICS Web Server plug-in, perform this action by using the IBM HTTP Server pwapi.so sample plug-in.

Application access to authentication data

If you have to perform security decisions within your CICS application code, a CICS Web support direct connection provides several ways to do this. The CICS WEB and TCPIP API allow you to access a lot of information about the client, including details about the TCP/IP address and SSL client certificate. This support is not available with the CICS Web Server plug-in, although the IBM HTTP Server does provide for limited decisions to be taken using the protection directives.

Sign-on support in CICS

Because an explicit EXEC CICS SIGNON or CESN transaction is not supported using CICS Web support, any existing logic that uses EXEC CICS SIGNON must be modified to use the CICS Web support authentication mechanisms, that is, authentication in the CICS Web support Analyzer with a direct connection or using the IBM HTTP Server with the CICS Web Server plug-in. If you cannot modify your application, consider using Host On Demand or the CICS Transaction Gateway external presentation interface (CTG EPI) interface.

5.3.2 CICS Web support security checklist

The following list is a summary of the key actions that you must consider if you use CICS Web support to Web-enable your CICS applications. (Note that most of these are normal CICS security considerations, and you may decide that you do not have to implement all these actions. However, it is recommended that you consider each one.)

- Set up one or more separate CICS Web-owning regions if you are using a CICS Web support direct connection, or use the CICS Web Server plug-in. A Web-owning region design is described in “Web-owning region” on page 173.

  The principal benefits of using a Web-owning region design are:

  - Isolation of the Web transactions into one CICS-specialized Web-owning region. Any failure in the Web-owning region, such as short-on-storage or excessive CPU consumption, does not affect your production regions. This also allows you to physically separate your traditional CICS applications into a separate AOR in order to prevent a rogue user application from accessing the storage or the information you do not want to be accessed.
  - The Web-owning region or the IBM HTTP Server may be placed in a different LPAR, perhaps between a filtering router and a firewall, that is, in a DMZ, in order to completely isolate any connected CICS regions from attack.

- The maximum security any Web user can have when running requests in the connected AOR can be limited by using CICS Link security, but not when using the 3270 Web bridge.
- Workload balancing techniques can be used to balance requests across multiple Web-owning regions and multiple AORs. CICSPlex System Manager (CICSPlex SM), TCP/IP port sharing, Dynamic Domain Name System (DNS), and the Load Balancer for Websphere Application Server can be used to balance such requests, and to limit the impact of denial-of-service attacks.

- Use a firewall or filtering router or both as the first line of defense. This allows you to restrict Internet access to a specific set of IP addresses and ports, to filter out some potential Internet-style attacks, and to hide the actual IP address that your Web server or CICS region is using.
- Use ports other than the default to avoid random attacks on known ports.
- Use a means of authenticating all the users who access the restricted information. There are several ways of doing this, including HTTP basic authentication, customized HTML form-based authentication, or SSL client certificates. This provides a first line of defense against attacks and also the ability to maintain an audit trail. If you are using HTTP basic authentication or HTML form-based authentication, use SSL to encrypt the user ID password flows.
- Do not transmit sensitive data as clear text across the Internet. Remember that this includes user IDs and passwords that are sent using HTTP basic authentication. SSL support within CICS or that provided with the IBM HTTP Server can be used to encrypt data.
- Be extremely careful when using the 3270 Web bridge. The supplied 3270 Web bridge allows Web access to any 3270-based transaction, greatly increasing potential security exposures. In any case, you must prevent Web access to transactions such as CEMT or CECI. This is best done by using CICS transaction security. However, it can be checked for in the Analyzer, or can be prevented by using a surrogate user ID with the CICS Web Server plug-in or a predetermined user ID specified in the Analyzer.
- Use a customized CICS Web support Analyzer when using a CICS Web support direct connection in order to restrict access to certain authenticated users, and transactions and programs. You can also have different TCPIPSERVICE definitions listening on different ports with different Analyzers. You must also be aware that by default:
  - The standard CICS-supplied Analyzer, DFHWBADX, which was recommended prior to CICS TS V3.1, allows specification of any alias transaction, converter program, and program.

  - The default Analyzer supplied with CICS TS V3.1 is DFHWBAAX. It does not extract alias attributes out of the URL, but assumes that you have installed URIMAP definitions.
  - The security sample Analyzer, DFH$WBSA, uses a token in the URL, which can be used in place of a user ID and password, by anyone with access to the URL sent to the Web browser.
- Implement resource security in addition to transaction-level security for all the programs in the Web-owning region. This prevents a program from being linked to by a malicious user by using a transaction such as the CECI.
- Ensure that the program autoinstall is turned off in the Web-owning region. Program autoinstall allows the loading of any program in the load libraries available to CICS, into CICS storage. Without a program definition, a CICS program cannot execute. You must also strictly control the programs in the load libraries available to the Web-owning region. If a program cannot be found, it cannot load and execute.
- Place limits on the maximum amount of system resources that can be used by transactions invoked from the Web. Following are the ways to do this:
  - Use the CICS TCPIPSERVICE BACKLOG parameter
  - Use the CICS transaction classes to limit the maximum number of active Web tasks, considering limiting alias, CWXN, and mirror transactions
  - Use the z/OS workload management restrictions on CPU usage for each address space such as TCP/IP, CICS, and Web server
  - Use the CICSPlex System Manager to allow workload management of CICS tasks
- Consider turning on the transaction isolation and storage protection. This protects the CICS region from wayward transactions, and prevents a rogue application from accessing storage areas not associated with its active task.

6

CICS Transaction Gateway

This chapter looks at the security issues that you face when using a service-oriented architecture (SOA) for your CICS applications, using the CICS Transaction Gateway (TG). This chapter discusses how the CICS TG addresses the issues of authentication, authorization, data integrity, and confidentiality. It also examines how CICS and the CICS TG work together with an External Security Manager such as Resource Access Control Facility (RACF) to provide these facilities.

You must decide whether you want to run the CICS TG on z/OS, or on a distributed platform such as a Windows® or a UNIX platform, and flow the request over a network to a remote CICS region. If you are using a distributed platform, you have the choice of whether to reuse the CICS 3270 interface by using the CICS TG external presentation interface (EPI) methods, or whether to interface with the CICS business logic using the CICS TG external call interface (ECI) methods.

Finally, the chapter summarizes the key security considerations you may face, and provides a security matrix and checklist pertaining to the issues raised in this chapter. Figure 6-1 shows an overview of possible CICS TG configurations.

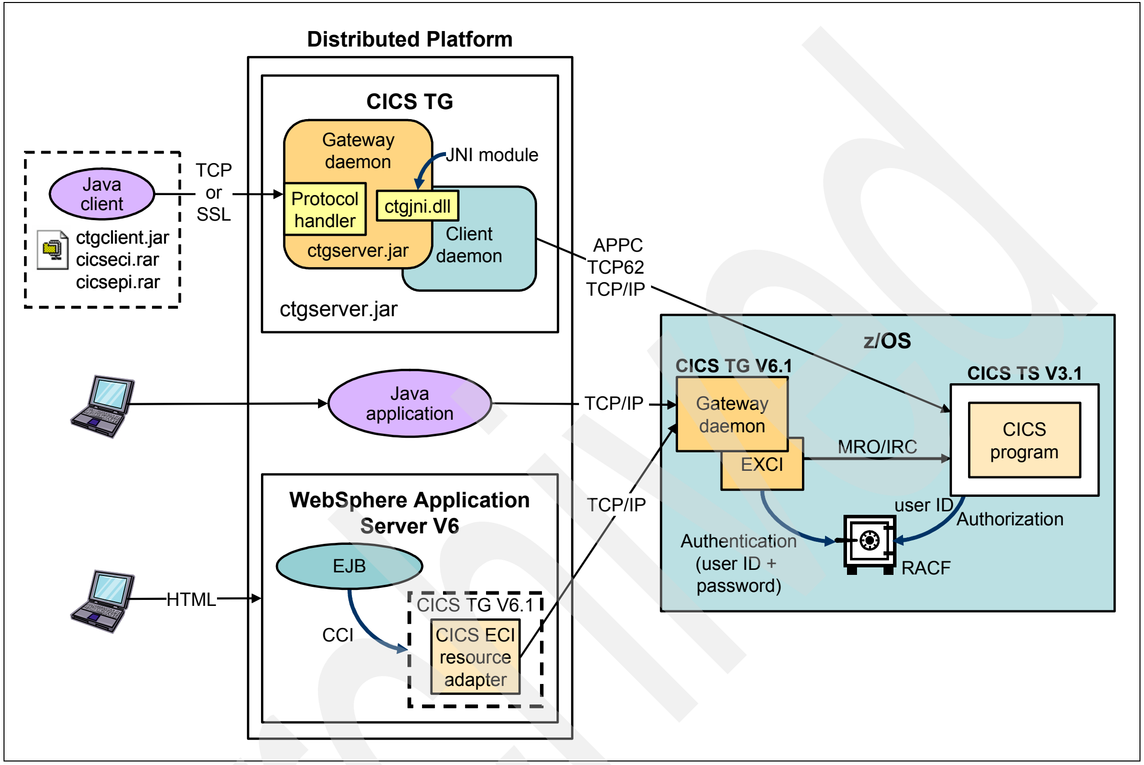


Figure 6-1 Overview of possible CICS TG configurations

6.1 Architecture choices

This section describes the security problems relating to the use of either a Java or a non-Java application with the CICS TG.

The CICS TG consists of a set of client and server software components that allow a remote client application to use services in a CICS region. The client application can be either a non-Java application using C, C++, COBOL, or COM interfaces (depending on the platform used) or a Java application.

When you use a Java application, the application can be any type of client, such as an applet, a servlet, or an enterprise bean. In the J2EE environment, the application is typically a servlet or enterprise bean that you deploy into a J2EE application server such as the IBM WebSphere Application Server.

CICS supports two main protocols for communication between a client program and a CICS application, EPI and ECI. With ECI, you can connect from the CICS TG to a CICS region over an IP network.

The Gateway daemon is a long-running process that functions as a server to network-attached Java client applications such as applets or remote applications by listening on a specified TCP/IP port. In local mode, there is no CICS TG daemon. In remote mode, there is a CICS TG daemon address space in z/OS that listens for ECI requests.

When you use the CICS TG on z/OS, only the ECI interface is supported. EXCI is a cross-memory implementation of ECI that is unique to z/OS. The most common z/OS configuration makes use of a local CICS TG. On z/OS, this results in a direct cross-memory EXCI connection between CICS and WebSphere Application Server.

J2EE connector architecture and security

The J2EE Connector architecture (JCA) has specific support for enabling secure access from a J2EE application to an enterprise information system (EIS) such as CICS. Both container-managed sign-on, in which the J2EE application server is responsible for flowing security context to the EIS) and component-managed sign-on in which the application is responsible for flowing security context to the EIS are supported.

In a managed J2EE environment, such as that provided by WebSphere Application Server, container-managed sign-on is recommended because it is good practice to separate the business logic of an application from qualities of service, such as security.

When deploying the component, the deployer must set the res-auth element in the deployment descriptor to indicate which method is being used. Figure 6-2 shows the flow for the JCA user ID selection.

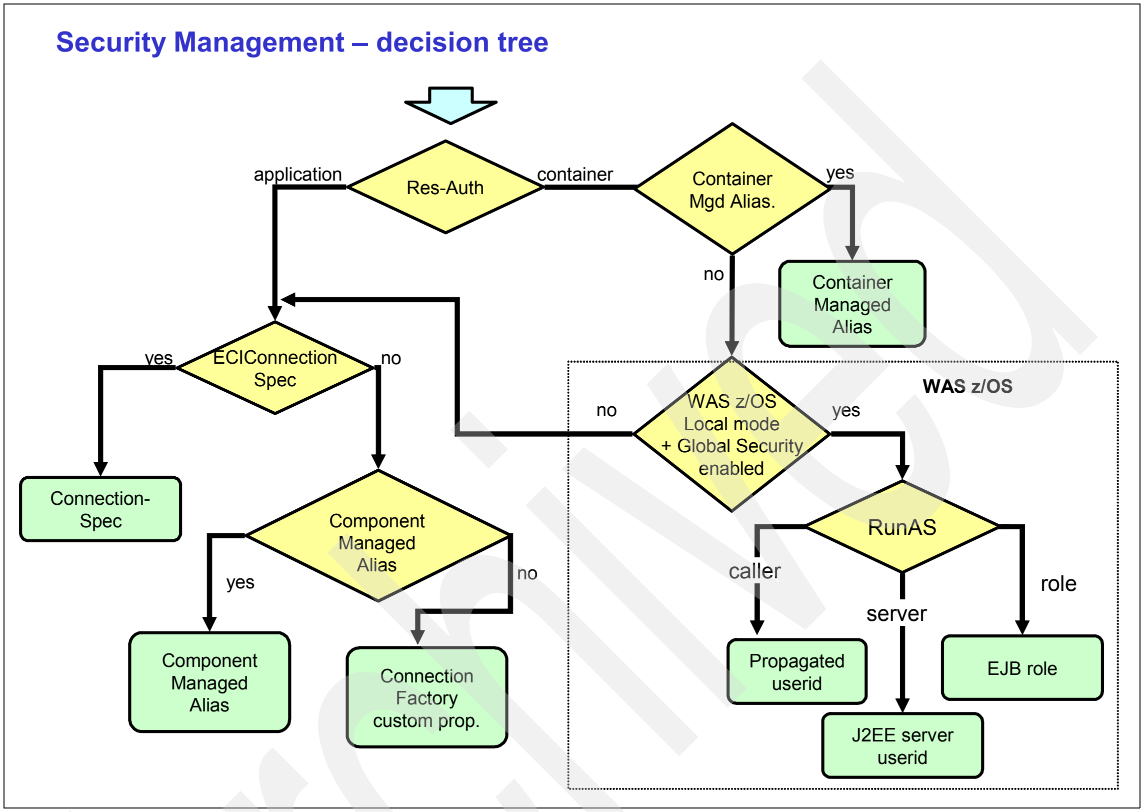


Figure 6-2 Flow sheet for JCA user ID selection

Container-managed security

If you are using container-managed security, you must set the res-auth deployment descriptor element to `Container`. The application deployer must set up the authentication information, for example, the deployer sets the family name (user ID) and password to be used for the connection. In certain circumstances, the container can derive the propagated identity from the currently executing Java principal. The application uses the getConnection()method of the connection factory and lets the application server manage the security to sign in to CICS.

A local mode configuration is one that follows the traditional z/OS authentication model, authenticating at the entry point to z/OS. For a J2EE application, it is normally in an HTTP server or in a WebSphere Application Server for z/OS. After authentication, your RACF user ID flows with you as you run work in different parts of the system. If the J2EE application connects to a CICS application, you must associate a RACF user ID with the request.

For connections using container-managed authentication and in local mode with a local registry configured, WebSphere Thread Identity support passes the user ID associated with the current Java thread in WebSphere Application Server for z/OS to the CICS ECI resource adapter and then to CICS.

For a local mode connection, CICS uses an EXCI connection, which allows a non-CICS address space on z/OS to communicate with CICS. EXCI assumes that the user ID is already authenticated for the user ID to be passed on to CICS without the password.

To avoid exposing the password, CICS allows only two options on the ATTACHSEC property, ATTACHSEC=LOCAL, where no user ID flows, or ATTACHSEC=IDENTIFY, where the user ID for the ECI request flows.

When using container-managed authentication, the WebSphere container is responsible for passing security information to the CICS ECI resource adapter. Known as thread identity, the security identity of the current Java thread is passed to the JCA connector and then to CICS. The user ID passed is the RunAs identity of the EJB that calls the connector. This could be:

- RunAs - Server, the application server's user ID
- RunAs - Caller, the identity of the caller of the EJB
- RunAs - Role, a user ID associated with a role

It is the absence of an association between the connection factory and a Java Authentication and Authorization Service (JAAS) authentication alias that enables thread identity support and causes the identity on the current thread to be passed to the CICS ECI resource adapter. Therefore, you must not define a JAAS authentication alias unless you want to prevent user ID propagation and instead, want to associate a fixed user ID (the JAAS alias) with all the requests to that CICS ECI connection factory.

JAAS authentication alias

JAAS is a package that enables services to authenticate and enforce access controls on users. JAAS is an optional part of Java 2 SDK 1.3, but is required by WebSphere Application Server.

If you want to use a JAAS authentication alias, and want to authenticate the user ID and password coded on the alias, tell the CICS ECI resource adapter to perform a user ID and password check by defining the AUTH_USERID_PASSWORD environment variable.

Component-managed security

For component-managed security, set the res-auth element to `Application`. The application code can then supply the user ID and password when making the connection. This is seen in the code sample shown in Figure 6-3. Note that even with res-auth=Application, the application can invoke a getConnection() without passing a user ID and password.

```
 Context ic = new InitialContext();

cnxf = (ConnectionFactory)     ic.lookup("java:comp/env/eis/ECICICS1");
// create a connectionSpec to hold the security information
ECIConnectionSpec cs = new ECIConnectionSpec();
// set the user ID/password
cs.setUserName(“user_ID”);
cs.setPassword(“password”);
Connection cxn = cxnf.getConnection(cs);
Interaction ixn = cxn.createInteraction();
ECIInteractionSpec ixnSpec = new
    ECIInteractionSpec(SYNC_SEND_RECEIVE,"CICSPROG");
JavaStringRecord jsr = new JavaStringRecord();
jsr.setText("DATA1");
ixn.execute(ixnSpec, jsr, jsr);
ixn.close();
cxn.close();
```

Figure 6-3 Component managed sign-on

Propagating an identity

This is perhaps the biggest problem facing most enterprises. If you are running WebSphere Application Server on z/OS, the problem is not so great because you can use the trusted identity support to flow the authenticated RACF security context from the WebSphere Application Server thread on to CICS by using the CICS TG, must be running in local mode.

However, if you are running WebSphere Application Server on multiplatforms, as many enterprises do, you have the problem of mapping a set of credentials, (perhaps certified using an operating system registry or a Lightweight Directory Access Protocol (LDAP) directory), to some trusted user in CICS. This is referred to as asserted identity in JCA. The problem is made easier if you are using the CICS TG on z/OS, which supports ATTACHSEC=IDENTIFY, which means the CICS TG can flow a user ID without a password. For more information about this, refer to “Determining the link user ID” on page 206. Note that in order to get an asserted identity to work for JCA, apply fix for APAR PK19503 for CICS TG for z/OS V6.1 because otherwise, the JCA resource adapter will not flow a user ID with a null password. However, the basic problem of how to assert an identity is the key one. Various solutions have been provided, including the following mechanisms:

- Use IBM Tivoli Access Manager and the global sign-on (GSO) Lockbox function.

  IBM provides the use of a GSO Lockbox through the Tivoli Access Manager plug-in for Web servers. The GSO Lockbox module provides access to applications, using the stored user credential information in the Principal Mapping Module, in order to map a credential to a RACF user ID.

  The GSO Lockbox is essentially a cache of user IDs that can be securely accessed with a client. However, the credentials have to be kept up-to-date manually, especially if supplying passwords. Figure 6-4 demonstrates the use of Tivoli Access Manager provided with WebSphere Application Server V6.

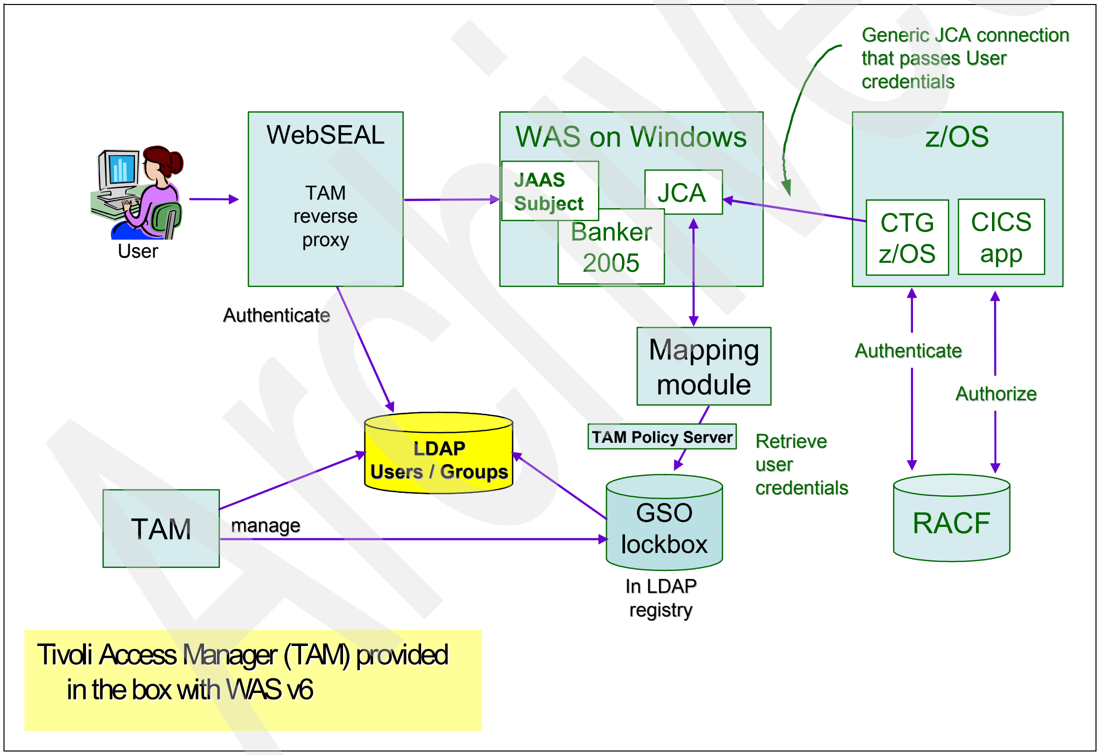


Figure 6-4 End-to-end propagation of user credentials

- Use CICS Link security either by using a Link user ID or with the Gateway region user ID to allow the security context to be asserted on a per Gateway basis

- Use SSL client certificate mapping to assert a user ID on a per client connection basis (see “Performing the encryption” on page 192)
- Develop your own mapping solution, perhaps by using RACF pass tickets

Performing the encryption

Most users want SSL direct in z/OS. This means that you must run the CICS TG on z/OS because this is the only topology that supports SSL for this purpose. Otherwise, you must use a virtual private network (VPN) from the CICS TG in CICS or some other kind of trusted network. In addition, there are the following points:

- You can map the Distinguished Name details in an SSL X509 client certificate to a RACF user ID, using the CICS TG RACFUserid class and the server side security exit.
- You can use SSL only for client authentication, and not payload encryption. To do this, you require a null cipher suite such as SSL_RSA_WITH_NULL_MD5, which has to be explicitly enabled in the CICS TG cipher suite list.
- You must think about CPU offload for SSL by using the IBM eServer zSeries hardware cryptographic support. Currently, the CICS TG only offloads handshakes when using CICS TG V6.x on z/OS by using Java Secure Sockets Extension (JSSE). Offloading handshakes to hardware substantially reduce the CPU costs.
- Think about how you want to control the SSL cipher suite. Many enterprises have mandates stating that only TDES or AES 256-bit can be used on the payload and only MD5 as a hash or RSA key sizes.

> **Note:** In JSSE, the cipher suite can only be controlled by the server (the CICS TG). Therefore, sometimes, the only way to get SSL clients to use a specific cipher is to have multiple CICS TGs because otherwise, all the SSL clients will negotiate up to the strongest cipher available.

6.2 CICS Transaction Gateway on distributed platforms

This section looks at CICS TG from a distributed perspective.

6.2.1 CICS TG for Multiplatforms V6.0

CICS TG for Multiplatforms V6.0 is supported on the following range of operating systems and platforms and is designed to support connectivity to all in-service CICS servers:

- Linux on System z
- Linux on Intel®
- Linux on POWER™
- AIX®
- HP-UX (on PA-RISC)
- Sun™ Solaris™ (on SPARC)
- Windows XP, Windows 2000, and Windows 2003

CICS TG for Multiplatforms comprises the following main runtime components:

- The Gateway daemon

  This listens for incoming work and manages the threads and connections necessary to ensure good performance.

- The Client daemon

  This provides the communication to CICS servers and the non-Java application programming interfaces (APIs).

- A Java class library or JCA resource adapter

  This is deployed into the client runtime environment. When used in a JCA environment, the resource adapter is deployed into the J2EE application server.

A Java client program can connect to a remote Gateway daemon using the TCP or SSL protocols. The Client daemon then provides the transport drivers to connect to the CICS server. Because the non-Java APIs are provided by the Client daemon, there is no remote connectivity support for non-Java clients.

6.2.2 CICS Transaction Gateway deployed on a distributed platform

In topology 1 (shown in Figure 6-5), both the WebSphere Application Server and the CICS TG are deployed on one of the distributed platforms, a Windows platform or a UNIX platform.

Figure 6-5 shows the authentication and authorization checks that are performed for an Enterprise JavaBeans (EJB) application that uses the ECI resource adapter to access a COMMAREA-based CICS application.

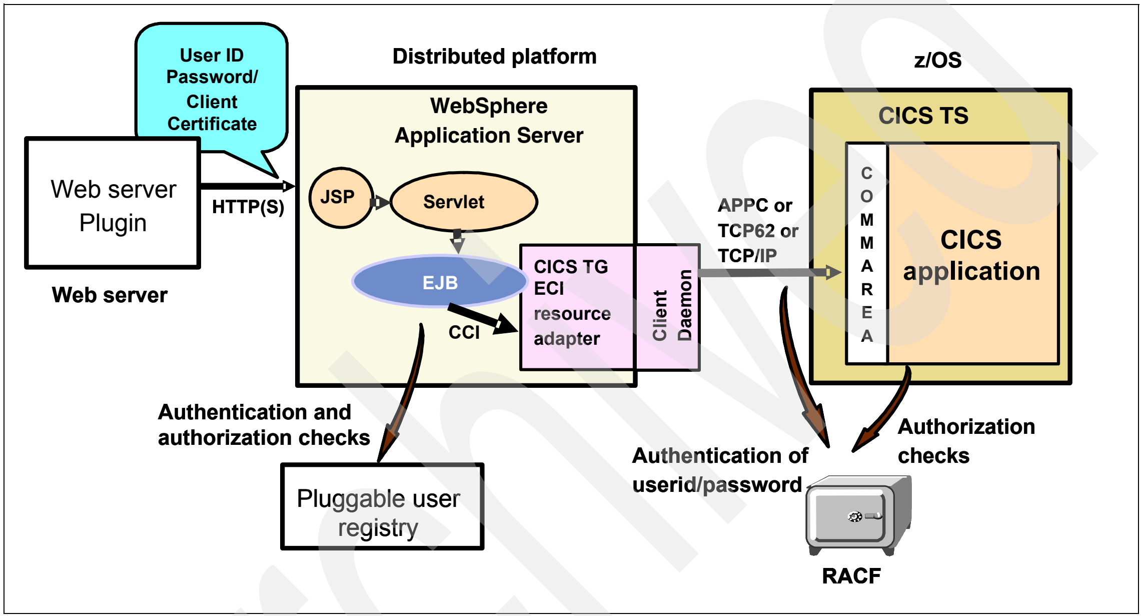


Figure 6-5 Authentication and authorization mechanisms for CICS TG: Topology 1

Authentication

An authentication mechanism in WebSphere Application Server typically collaborates closely with a pluggable user registry when performing authentication. The user registry allows you to configure different databases to store user IDs and passwords that are used for authentication and authorization.

Following are the user registry options:

- Local operating system user registry

  When configured, the application server uses the operating system's users and groups for authentication.

- LDAP user registry

  In many solutions, an LDAP user registry is recommended as the best solution for large-scale Web implementations. Most of the LDAP servers that are available in the market are well equipped with security mechanisms that can be used to securely communicate with WebSphere Application Server.

- Custom user registry

  This is the option for any custom implementation of a user registry database. An application server API provides the user registry Java interface that can be used to write the custom registry. This interface can be used to access virtually any relational database, flat files, and so on.

The authentication mechanism is responsible for creating a credential, which is an application server internal representation of a successfully authenticated client user. WebSphere Application Server provides two authentication mechanisms:

- Lightweight Third Party Authentication (LTPA)

  LTPA is intended for use with multiple application servers and machine environments. It supports the forwarding of credentials and single sign-on.

- Simple WebSphere Authentication Mechanism (SWAM)

  SWAM is intended for simple, nondistributed, application server configurations and is less secure than the LTPA.

Authorization

Pluggable authorization interfaces allow the use of different authorization mechanisms for WebSphere applications. WebSphere Application Server standard authorization mechanisms are based on the J2EE security specification and JAAS.

The following steps describe the main authentication and authorization events from the point when a Web browser sends a request to the WebSphere application:

1. The Web user requests a Web resource protected by the WebSphere Application Server
2. The Web server receives the request, recognizes that the requested resource is on the application server, and using the Web server plug-in, redirects the request.
3. The Web server plug-in passes the user credentials to the Web container of the application server, which performs user authentication against the user registry.

4. After successful authentication, the Web container performs authorization checks against the user registry of the user's credentials and the security information contained in the deployment descriptor.

5. On subsequent requests, further authorization checks are performed either by the Web container or the EJB container with user credentials that are extracted from the established security context.

6. When the EJB uses the ECI resource adapter to make a request to the CICS application, the security credentials (user ID and password) must be propagated through to CICS. This can be the responsibility of the application (component-managed sign-on) or it can be the responsibility of the Web or EJB container (container-managed sign-on).

   For both the container-managed sign-on and component-managed sign-on, the principal means of enabling authentication is by specifying a predefined security credential known as the JAAS authentication alias.

7. After defining the JAAS authentication alias, it can be associated with a particular connection to CICS when it is selected from a drop-down box when defining the connection factory. Another way of determining which user ID and password combination is propagated to CICS is to specify these in an ECIConnectionSpec when the connection is created.

8. When the request arrives in CICS, the user ID and password combination is verified against the RACF database. The CICS CONNECTION or TCPIPSERVICE definition must be specified with ATTACHSEC=VERIFY.

   This means that there must be some form of mapping between the user IDs stored in the user registry used by the application server and the user IDs stored in the RACF.

9. After successful authentication by CICS, the CICS application runs and CICS resource authorization checking is performed against the flowed user ID (the user ID that is specified in the JAAS authentication alias). The same checks are also performed against the link user ID if it is specified in the CICS SESSIONS definition. Note that the option of using a link user ID is *not* available when a TCP/IP connection to CICS is used.

> **Note:** To change the user ID and password information held in the CICS external security manager (ESM), use the ESI, which is based in the CICS password expiration management (PEM) function. There is no JCA resource adapter support for the ESI. Only the CICS TG base classes provide support.

Data integrity and confidentiality

Figure 6-5 shows that Hypertext Transfer Protocol Secure (HTTPS) can be used to secure the link between the Web server and the WebSphere Application Server. The Java Secure Socket Extension (JSEE) is the SSL implementation used by the WebSphere Application Server. It is a set of Java packages that enable secure Internet communications. It implements a Java version of SSL and Transport Layer Security (TLS) protocols and includes functionality for data encryption, message integrity, server authentication, and client authentication.

However, the link between the application server and CICS cannot be secured using SSL, that is, the user ID and password are not encrypted. The security of this link is, therefore, dependent on the security offered by the physical configuration.

6.3 CICS Transaction Gateway on z/Series

This section focuses on CICS TG on the zSeries platform.

6.3.1 Remote Gateway daemon on z/OS

In topology 2 (shown inFigure 6-6), where WebSphere Application Server is deployed on one of the distributed platforms, access to CICS is through a Gateway daemon running on z/OS, as shown in Figure 6-6.

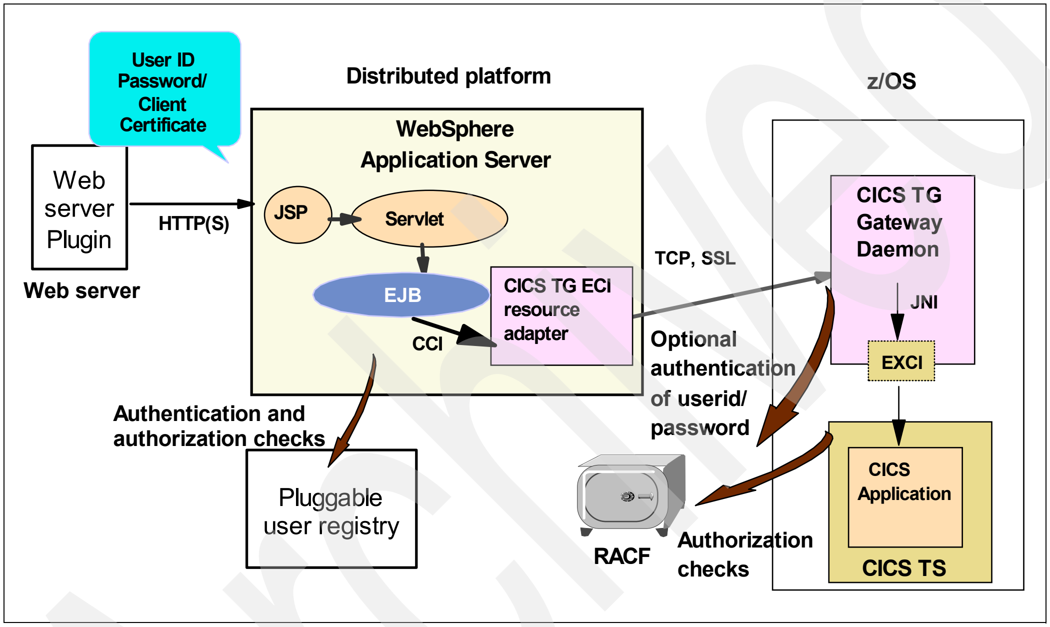


Figure 6-6 Authentication and authorization mechanisms for CICS TG: Topology 2

There are several important security differences between this topology and topology 1 shown in Figure 6-5 on page 194.

Authentication

The same application server authentication options that were discussed for topology 1 apply equally to topology 2. There is, however, an important difference in the authentication processing on z/OS.

Topology 1 requires that a user ID and password are flowed with each ECI request. This can be inconvenient in situations where authentication is being undertaken using a mechanism other than user ID and password authentication, such as client certificate authentication. In such situations, user ID authentication by CICS does not easily fit into the overall security design of the solution. Using topology 2 can help you to avoid this problem because CICS TG for z/OS allows a preauthenticated user ID to be flowed into CICS without a password. The authentication of user ID and password in this topology is optional.

To enable the CICS TG to authenticate each user ID and password flowed on an ECI request, the environment variable AUTH_USERID_PASSWORD must be set in the CICS TG environment variables. If user ID and password checking is not performed, it will probably be necessary to devise a way to establish a trust relationship between the application server and the Gateway daemon, so that the application server can be trusted to flow only the user ID on the request through to CICS by using the Gateway daemon. Solutions such as SSL client authentication and virtual private networks (VPN) can be used to establish such a trust relationship.

Because no password is flowed to CICS when you use the CICS TG on z/OS, the EXCI CONNECTION definition must be defined with ATTACHSEC=IDENTIFY. IDENTIFY means that CICS uses the flowed user ID in the EXCI request, but does not expect a password to be flowed with the request because this is (optionally) checked by the CICS TG.

Authorization

In addition to the authorization checks described for topology 1, additional authorization checks can be used when the CICS TG is deployed on z/OS. These include:

- Multiregion operation (MRO) bind security

  Use MRO bind security to prevent unauthorized attached MRO regions from starting transactions in a CICS region. Use the RACF DFHAPPL profiles in the FACILITY class to control the login to DFHIRP. This determines whether a particular CICS TG can connect to a CICS region.

- Link security

  The link user ID that is used for authorization checks in CICS is the user ID that is associated with the started task of the CICS TG Gateway daemon. Because this is likely to remain the same after the initial configuration, it can be preset in the EXCI SESSIONS definition.

- Surrogate security

  Use surrogate security to authorize the user ID that is associated with the CICS TG-started task to switch the security context of an EXCI request to the flowed user ID. Control surrogate security by using a profile in the SURROGAT class of RACF.

Data integrity and confidentiality

The CICS TG for z/OS provides SSL support by using the JSSE. Following are some of the features of JSSE on z/OS:

- RACF key ring support

  SSL key stores can now be stored in a RACF database
- zSeries hardware cryptographic support

  This provides the ability for the CPU to offload SSL handshakes to hardware. This can substantially reduce the CPU cost of SSL handshakes and SSL data encryption.
- SSL cipher suite selection

  The SSL cipher suite that is in use can be configured. When using SSL with this topology, it is particularly important to have an efficient connection pooling mechanism because otherwise, a significant proportion of the time - from making the connection to receiving the result from CICS and closing the connection - can be in the SSL handshaking. The JCA connection pooling mechanism mitigates this overhead by allowing connections to be pooled by the WebSphere Application Server pool manager so that SSL handshaking is not required for each request.

Note: The link between CICS TG and CICS TS in this topology is MRO (Cross Memory). Therefore, cryptography is not necessary.

6.3.2 WebSphere Application Server and CICS Transaction Gateway on zSeries

In a zSeries topology, WebSphere Application Server can be deployed on either a z/OS system or on a Linux operating system. The security mechanisms differ significantly between these two topologies.

WebSphere Application Server and CICS Transaction Gateway on z/OS

This topology (shown in Figure 6-7) has significant security advantages for the following reasons:

- The application server and CICS are able to share the same RACF user registry for authentication and authorization checks (see Figure 6-7).
- The application server and CICS are installed in the same Multiple Virtual Storage logical partition (MVS LPAR) and therefore, the connection between the servers is inherently more secure.

- Thread identity support is enabled (see “Authorization” on page 201).

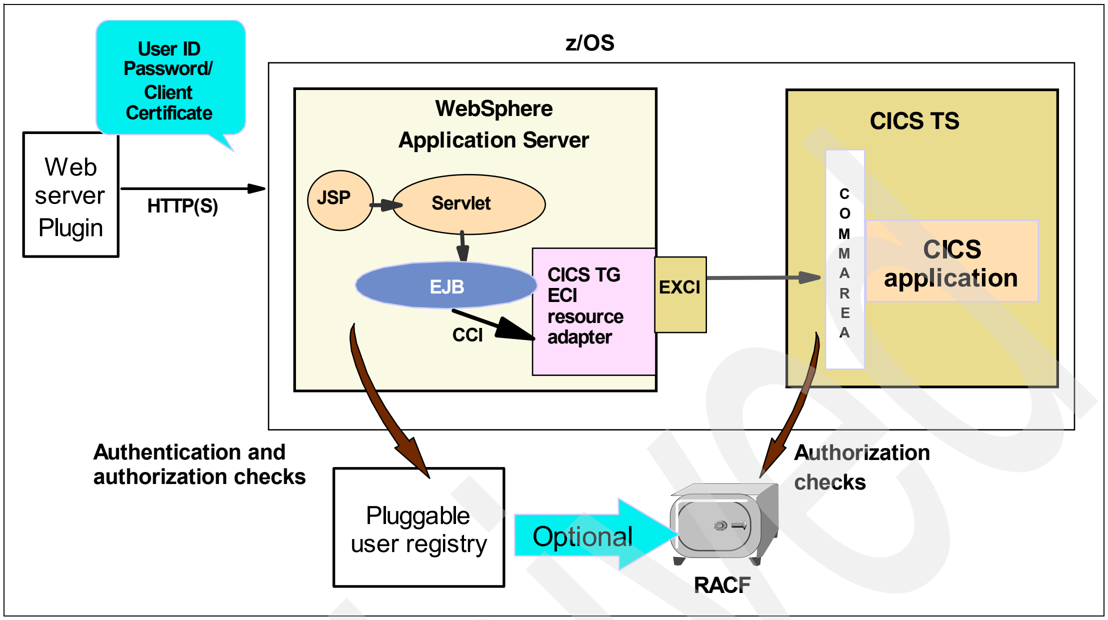


Figure 6-7 Authentication and authorization mechanisms for CICS TG: Topology 3a

Authentication

When WebSphere Application Server is running on z/OS, the same options for a pluggable user registry apply, that is, a local OS registry, an LDAP registry, or a custom user registry. When the application server is configured to use a local OS registry such as RACF, the security identity established after authentication in WebSphere Application Server is a RACF user ID if you use basic authentication or form-based login. If an SSL client certificate is used to authenticate, you can configure RACF to map that certificate to a RACF user ID. This means that the Java thread in WebSphere Application Server on z/OS will have a security identity that is a RACF user ID.

Authorization

When using container-managed sign-on, a z/OS system-specific functionality known as *thread identity support* is provided by WebSphere Application Server for z/OS. This support is unique to WebSphere Application Server for z/OS and allows the application server to automatically pass the user ID of the thread (the caller's identity) to CICS when using the ECI resource adapter.

Thread identity support is enabled under the following situations:

- WebSphere Global security is enabled and RACF is being used as the local OS registry
- A local connection is being used between the application server and CICS
- Container-managed security is being used (the res-auth deployment descriptor is set to `Container`)
- `The` connection factory does not specify a JAAS authentication alias

Most z/OS customers want to use this feature because it enables the application server to behave in a way that traditional z/OS address spaces behave, that is, after you have authenticated, your user ID flows with any work you do within the z/OS system.

The CICS authorization mechanisms apply to this topology as follows:

- MRO bind security

  Use MRO bind security to establish a trust relationship between the application server and CICS servers. Use the RACF DFHAPPL profiles in the FACILITY class to control the login to DFHIRP.

- Link security

  The link user ID that is used for authorization checks in CICS is the user ID that is associated with the started task of the WebSphere J2EE servant region. Because this is likely to remain the same after the initial configuration, this can be preset in the EXCI SESSIONS definition.

- Surrogate security

  Enable surrogate security checks to authorize the user ID associated with the J2EE servant region to flow a specific user ID or one of a generic set of user IDs to CICS.

Data integrity and confidentiality

WebSphere Application Server for z/OS supports JSSE as the SSL service provider for the Web and EJB containers. JSSE is a pure Java implementation that is common across all IBM platforms. IBMJSSE implements SSL 3.0 and TLS 1.0 algorithm types as Java 2 standard extensions. IBMJSSE is the preferred SSL and TLS provider for Java applications on z/OS, and must be used in place of SystemSSL for Java applications. Because the link between the application server and CICS is normally a local cross-memory link, it is not necessary to encrypt the data that is passed between the application server and CICS.

For more information about managing security in this z/OS topology, refer to *z/OS WebSphere Application Server V5 and J2EE 1.3 Security Handbook,* SG24-6086*.*

WebSphere Application Server and CICS TG on Linux on System z

In this topology (shown in Figure 6-8), the WebSphere Application Server is deployed within Linux on System z. The security options for this topology are almost identical to those described for topology 1. One notable exception is that in this topology, you can use HiperSockets™ to connect from the CICS TG running on Linux to the CICS server. Therefore, it is unlikely that you have to encrypt the data that is passed on this link.

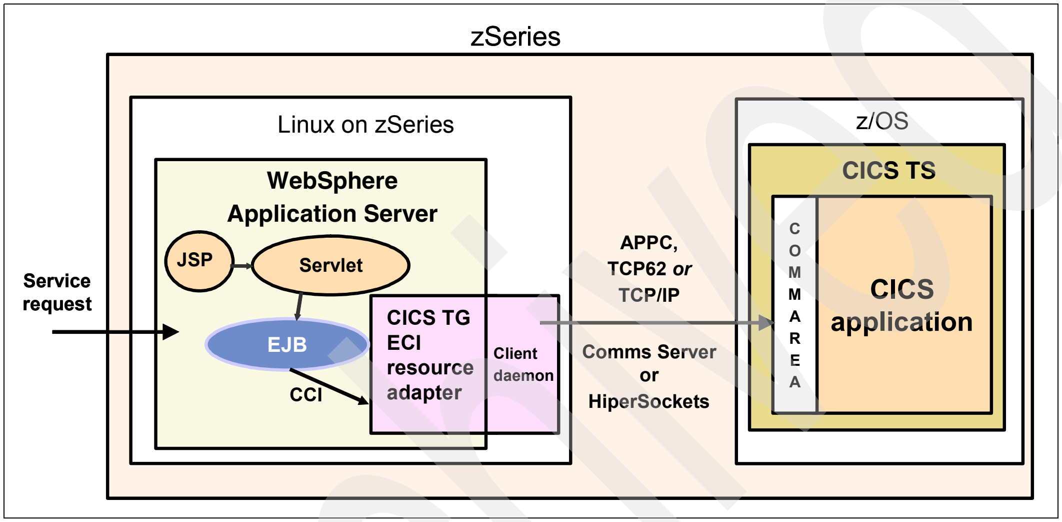


Figure 6-8 CICS TG: Topology 3b

6.3.3 Security coordination between WebSphere and CICS

This section summarizes the key security points that you have to understand when using the JCA to connect to CICS from a J2EE application running in the WebSphere Application Server:

- You have a choice of using container-managed sign-on or component-managed sign-on. Container-managed sign-on is the recommended approach.
- When using the ECI resource adapter, the support provided by the JCA security management contract is dependent on the CICS TG topology being used.
- Defining a JAAS authentication alias is the principal way of specifying the user ID and password to be flowed to CICS when using topologies 1 and 2 (WebSphere Application Server installed on a distributed platform).

- When using the ECI resource adapter with topology 1 (CICS TG installed on a distributed platform), a user ID and password must be flowed with each ECI request. In contrast, you have the option of flowing only the user ID when using topology 2 (CICS TG installed on z/OS).
- When using the ECI resource adapter with topology 3a (WebSphere Application Server installed on z/OS) the application server is capable of automatically passing the caller's authenticated user ID to CICS.
- You can use a number of CICS security mechanisms, including bind, link, and surrogate security to provide additional security checks.
- You can use SSL in most configurations if there is a requirement to encrypt the data that is passed between the different physical tiers in the topology.

6.4 CICS Transaction Gateway for z/OS V6.1

CICS TG for z/OS V6.1 is the latest version of the z/OS product. It is supported on z/OS V1R4 and later, and supports connectivity to CICS TS for z/OS V1.3, V2.2, V2.3, and V3.1.

CICS TG for z/OS uses the external communication interface (EXCI) provided by CICS TS to communicate with CICS. It does *not* include the Client daemon and does *not* provide any support for non-Java based applications because this support is provided through the CICS EXCI interface.

For all CICS TG topologies, there is a requirement to authenticate the user and to ensure that only authorized users have access to the application and its resources. The CICS TG uses the z/OS System Authorization Facility (SAF) to route authorization requests to an external security manager (ESM) to perform all its security checks. You can use any suitable ESM. However, because the RACF product from IBM is the most commonly used, the rest of this book refers to it. For complete information about CICS security, refer to the *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454.

For information about configuring a secure CICS TG for a z/OS environment, refer to *CICS Transaction Gateway for z/OS Version 6.1*, SG24-7161.

External call interface using COMMAREA applications

The CICS TG on z/OS only supports the ECI. The ECI allows a non-CICS application to call a CICS program in a CICS server. The CICS TG does *not* provide an EPI interface and so does *not* provide support for 3270-based transactions.

Determining the flowed user ID

The flowed user ID is the identity that is passed on the ECI request. It often represents the identity of the application user. When using the base CICS TG classes, the user ID and password can be specified as parameters when constructing an ECIRequest object.

Because the CICS TG runs as a shell process under UNIX System Services, any user ID that it tries to authenticate with RACF, such as a user ID flowed with an ECI request, must have an OMVS segment defined in its RACF profile. For more information about this, refer to *z/OS V1R8.0 UNIX System Services Planning*, GA22-7800.

If you receive a message such as “`CTG6808E Authorize userid/password with RACF. User ID = CICSCTG , Return code = -1, errno = 157 , errno2 = 0X90C02AF`”, refer to *z/OS V1R8.0 UNIX System Services Messages and Codes*, SA22-7807 for the meaning of “errno”.

The errnos are, for example, listed by value in Section 3.0. An errno of 157 indicates that a MVS environmental or internal error has occurred. The meaning of errno2 can be found in 5.2 Reason Codes. The last 4 digits of errno2 is 02AF indicating JREnvDirty. The specified function is *not* supported in an address space where a load that is not program controlled was done. The action to be undertaken is to make sure that the programs being loaded from this address space are defined as program-controlled.

Determining the mirror transactions

An EXCI request received by CICS from the Gateway daemon attaches a mirror transaction. Therefore, you must authorize your flowed user ID and your link user ID to your private mirror transaction.

> **Tip:** In a production environment, you may create a group of users who require common access. When you build a group, you permit access to a group so that user access can be controlled by the group to which a user belongs, rather than by individual permissions. This method simplifies the security definitions that are required.

Determining the link user ID

The link user ID is the user ID associated with the Gateway daemon that is passing the request to CICS. If the link security is enabled, the link user ID, like the flowed user ID, must be authorized to access all the transactions and resources used as a result of the request.

The link user ID can be preset in the EXCI SESSIONS definition using the USERID parameter. For EXCI requests, the link security works as follows:

1. If the link user ID is the same as the CICS region user ID, the systems are deemed *equivalent* and no link security authorization is performed.
2. If the link user ID is preset as something other than the CICS region user ID, this user ID must be authorized to access all the transactions and resources invoked as a result of the request.
3. If the link user ID is not preset, the user ID of the connected region, that is, the user ID of the Gateway daemon started task, is the link user ID, and this user ID must be authorized to access all the transactions and resources invoked as a result of the request.

When running any application that connects to CICS through the EXCI, such as the Gateway daemon, CICS writes a DFHSN1400 Session sign-on message, followed by a DFHSN1500 Session sign-off message pair to the CICS joblog for each request. By default, this message pair is written for each ECI request. However, when the link user ID is preset, EXCI sign-on is performed when the CICS CONNECTION is installed, but not for each ECI request.

It is also possible to suppress EXCI sign-on messages by using the CICS user exit XMEOUT. An assembler sample of this program can be found in SDFHSAMP(DFH$SXP1).

It is recommended that you specify `ATTACHSEC=IDENTIFY` and use nonequivalent systems, so that security checks are also performed against a a link user ID.

Table 6-1 lists the different settings for link security and ATTACHSEC and how they interoperate.

Table 6-1 Attach security settings with an EXCI connection from the Gateway daemon

Equivalent systems	**Link user ID = CICS region user ID**		**Link user ID not = CICS region user ID**	
ATTACHSEC	**LOCAL**	**IDENTIFY**	**LOCAL**	**IDENTIFY**
Link user ID check	NO	NO	YES	YES
Flowed user ID check	NO	YES	NO	YES

Equivalent systems	Link user ID = CICS region user ID		Link user ID not = CICS region user ID	
ATTACHSEC	**LOCAL**	**IDENTIFY**	**LOCAL**	**IDENTIFY**
User ID mirror transaction runs under in CICS	CICS default user ID	Flowed user ID	Link user ID	Flowed user ID

6.4.1 Sample security programs

The CICS TG provides several sample programs to assist you in setting up security with your CICS TG region. There are samples that run as batch jobs under z/OS, and others that run as Java programs.

The job CTGTESTR is supplied in the SCTGSAMP library. CTGTESTR runs the EciB1 sample through an active CICS TG daemon and checks the connectivity to the first CICS Transaction Server index defined. The JCL passes the URL, port, user ID, and password to the CICS TG. If the CICS TG has a working connection to a backend CICS region, the request flows across an EXCI link and runs program EC01 (Figure 6-9).

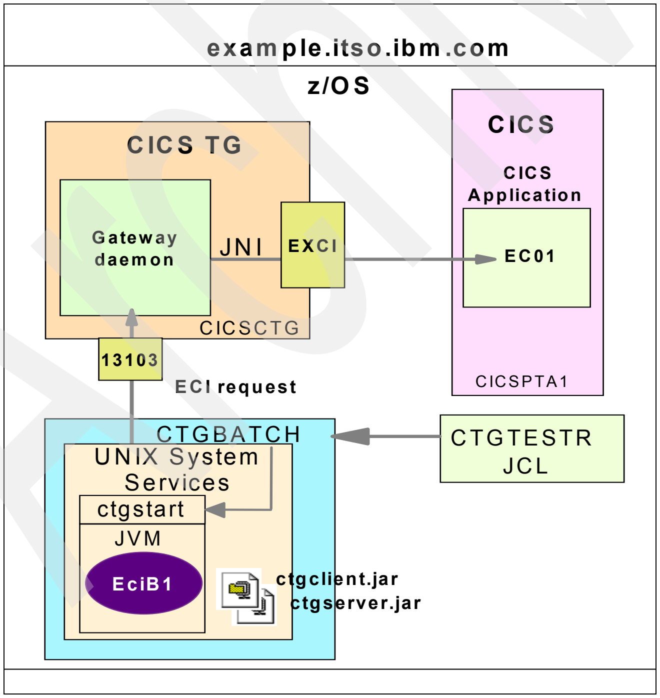


Figure 6-9 Test scenario for the CTGTESTR JCL

The EC01 program returns the date and time that is presented in the CTGTESTR JCL in hex and ASCII format.

A Java program called EciB1 is a sample program to test that the CICS TG user ID password authentication is working properly. This program is in the ctgsamples.jar supplied in the classes directory. The source for this program is provided in the \samples\java\com\ibm\ctg\samples\j2ee directory.

The sample TestECI program earlier provided by the CICS TG has been replaced by the EciB2 sample program. This program allows you to control the input parameters from the command line.

6.4.2 Secure Sockets Layer

SSL is a protocol designed to create a secure connection to a server, using public key encryption, and to protect the data integrity and privacy as the data is transferred over the connection. CICS TG supports the JSSE implementation of SSL, which provides 128-bit encryption and 256-bit encryption. JSSE is supplied as part of the IBM SDK for z/OS, Java 2 Technology Edition, V1.4.2 SR2.

In the earlier versions of CICS TG, you were able to use SystemSSL, which is specific to z/OS, and the Java-based SSLight. JSSE is the only supported option for providing SSL with CICS TG V6 and later releases.

> **Important:** SystemSSL and SSLight are *not* supported by CICS TG V6 and later releases.

The SSL Handshake Protocol consists of two phases:

1. Server authentication

   In the first phase, the server responds to a client’s request by sending its certificate and cipher preferences. The client then generates a master key, which it encrypts with the server’s public key, and then transmits the encrypted master key to the server. The server authenticates itself to the client by returning a message authenticated with the keys derived from the master key. Subsequent data is encrypted and authenticated with keys derived from the master key.

2. Client authentication

   In the second optional phase, the server requests that a client identifies itself during the SSL handshake by providing its client certificate. Client authentication can only be requested by the server.

   > **Note:** CICS TG supports server authentication and client authentication.

When setting up an SSL environment, you have a choice between self-signed certificates and CA-signed certificates:

- Self-signed certificates

  A self-signed certificate is an identity certificate that is signed by its creator. In this situation, the creator is verifying that the certificate is valid.

- Certificating authority (CA) signed certificates

  CA-signed certificates are created by a user organization and sent to a CA to be signed. Before signing a certificate, the CA verifies that the organization requesting the certificate is actually who they claim to be. In this situation, it is the CA who is verifying that the certificate is valid.

  In a production environment, the CA-signed certificates are expected to be used because they provide a more secure solution.

6.4.3 Java Secure Sockets Extension

JSSE is a Java implementation of SSL common across all platforms. JSSE implements SSL 3.0 and Transport Layer Security (TLS) 1.0 algorithm types as Java 2 standard extensions. It provides the socket factories that allow for ease of programming if the provided defaults are taken.

JSSE is now the strategic SSL implementation for CICS TG and supports the following:

- Cryptographic handshakes on z/OS using an IBMJCE4758 Peripheral Component Interconnect (PCI) card
- Management tools to generate keystores and manage certificates. These include:
  - keytool

    A command-line interface for maintaining keystores
  - hwkeytool

    A command-line interface for maintaining hardware keys
  - ikeyman

    A graphical user interface (GUI) for maintaining keystores
- The ability to store digital certificates in a RACF database.

Restriction: At this point in time, CICS TG does *not* support TLS.

6.4.4 Cipher suites

IBM SDK 1.4.2 JSSE supports a wide range of cipher suites. Example 6-1 provides an explanation of the cipher suite naming convention.

Example 6-1 Cipher suite components

```
SSL_RSA_WITH_RC4_128_SHA

RSA is a type of key exchange cipher
RC4_128 is the cipher for data encryption
SHA is the hashing algorithm
```

The *key exchange cipher* is used for the initial handshake. The most widely used key exchange cipher is Rivest-Shamir-Adleman (RSA). This type of cipher is referred to as asymmetric because it uses the public key and the private key.

The *cipher for data encryption*, also known as the data encryption cipher, is used to secure the data that is passed over the SSL connection after the handshake is completed. Possible data encryption ciphers are Advanced Encryption Standard (AES), Data Encryption Standard (DES), Triple DES, RC2 (“RC” stands for “Ron’s Code” or “Rivest’s Cipher”), and RC4, which have 40-bit,128-bit, and 256-bit variants. These ciphers are referred to as symmetric because they use a single key for encryption and decryption. Symmetric key or secret key algorithms can be further classified as either *block ciphers* or *stream ciphers*. Block ciphers, such as DES, operate on the data in blocks and stream ciphers, such as RC4, operate on the data one bit at a time.

The *hashing algorithm* is a one-way method of producing a hash number from a message. A slight change in the message produces a completely different hash number (a 128-bit fingerprint) and is used to ensure that a message has not been tampered with. The client and the server both generate a hash number from the data transmitted using the hashing algorithm. If the hash numbers do not match, it means that the integrity of the message has been compromised. Data integrity is provided by one of the following hashing algorithms, also known as message digest algorithms, SHA1 (Secure Hash Algorithm), SHA2, SHA3, SHA5, MD2, and MD5. Cipher suite names ending in _SHA indicate that the SHA1 hashing algorithm is being used.

SSL can use signature algorithms SHA1 with RSA, SHA1 with DSA, MD5 with RSA, and MD2 with RSA.

Tips: CICS TG V6 supports a range of different cipher suites and allows you to specify which ciphers to use. When deciding this, you must consider the following performance criteria:

- You can significantly reduce the number of SSL handshakes when connecting from the WebSphere Application Server by using JCA connection pooling.
- RSA encryption and decryption processing during the SSL handshake can be offloaded using crypto engines, PCICA, PCIXCC, or CEX2C.
- Larger keys (512/1024/2048) for key exchange ciphers have a higher handshake cost.
- Choose appropriate bulk data ciphers. It is generally accepted that:
  - MD5 is faster than SHA, but is less secure
  - RC4 normally has a lower CPU cost than DES in software
  - RC4 stronger ciphers (128-bit versus 40-bit) do not require more CPU cycles
  - SHA, DES, and AES can often be implemented efficiently in hardware

Resource Access Control Facility key ring

To create a RACF key ring, you can either migrate the certificates that you have already built in a key database with the *gskkyman* utility, or you can generate new certificates using the RACF RACDCERT command. In z/OS, RACF is the tool of choice because it provides known, trusted, and auditable access, and a secure database for credentials.

If you are creating your own self-signed certificate, you may want to use the CICS-supplied sample REXX exec called DFH$RING. This creates several certificates and a RACF key ring, and then adds the new certificates and the CA certificates to the key ring. DFH$RING is supplied in the SDFHSAMP data set.

For information about creating a key ring and certificates on z/OS using RACF, refer to *CICS Transaction Gateway for z/OS V6.1,* SG24-7161 .

6.5 Designing a secure solution

This section first provides a decision matrix to summarize the security support of the CICS TG. It the provides a checklist of implementation tips that you must consider when using the CICS TG to access CICS applications.

6.5.1 CICS Transaction Gateway security matrix

The security matrix in Table 6-2 summarizes the key architectural questions for securing access to CICS, using either CICS TG on z/OS or on a distributed platform. Each item is discussed in detail in the subsequent sections.

Table 6-2 CICS TG security decision matrix

Architectural questions	Distributed CICS TG	CICS TG on z/OS
Authentication of client by user ID and password	Flow the user ID on an ESI, EPI, or ECI request	Use a preauthenticated user ID or set the AUTH_USERID_PASSWORD environment variable
SSL encryption support	Yes, CICS TG supports the JSSE implementation of SSL	Yes, CICS TG supports the JSSE implementation of SSL, which provides 128-bit and 256-bit encryption
SSL client certificate support	Yes, using client authentication during the SSL handshake	
Flowing user ID into CICS	Yes, use ATTACHSEC=VERIFY to flow a user ID and password for authentication to CICS	Yes, use ATTACHSEC=IDENTIFY to flow user ID into CICS
Handling unregistered client certificates	RACF certificate name filtering feature	
Mapping of many users to one RACF user ID	RACF certificate name filtering feature	
Restricting maximum user authorization	Yes, restrict the link user ID on the connection into CICS	
Handling expired password	Use the ESI to check and manage passwords. Can be used only with CICS servers that support password expiry management (PEM).	Use Password Application Programming Interface (PWAPI) plug-in or change with HTTP basic authentication prompts or use RACF Java API
Application access to authentication data	No, but the CICS TG security exit can access authenticated data	
Sign-on support in CICS	Sign-on possible by using the EPIRequest class	Not required because attach security is used with the ECI to authenticate the user in CICS TG

Authentication of client by user ID and password

When using the CICS TG on a distributed platform, the user ID and password are flowed with each request. When the CICS TG is on z/OS, you can flow a preauthenticated user ID into CICS without a password. The authentication of the user ID and the password in this topology is optional.

To enable the CICS TG to authenticate each user ID and password flowed on an ECI request, the variable AUTH_USERID_PASSWORD must be set in the CICS TG environment variables.

Secure Sockets Layer encryption support

SSL encryption creates a secure connection to a server using public key encryption and protects the data integrity and privacy as the data is transferred over the connection. The CICS TG supports the JSSE implementation of SSL that provides 128-bit and 256-bit encryption. JSSE is supplied as part of the IBM SDK for z/OS Java 2 Technology Edition, V1.4.2 SR2.

Secure Sockets Layer client certificate support

The SSL Handshake Protocol consists of two phases:

1. Server authentication

   In the first phase, the server responds to a client's request by sending its certificate and cipher preferences. The client then generates a master key that it encrypts with the server's public key and then transmits the encrypted master key to the server. The server authenticates itself to the client by returning a message authenticated with keys derived from the master key. Subsequent data is encrypted and authenticated with keys derived from the master key.

2. Client authentication

   In the second optional phase, the server requests that a client identify itself during the SSL handshake by providing its client certificate. Client authentication can only be requested by the server.

> **Note:** CICS TG supports both server and client authentication.

Flowing user ID into CICS

You many want to flow a user ID into CICS when using the CICS TG to authenticate each request. When using the CICS TG on a distributed platform, specify `ATTACHSEC=VERIFY` in the CICS CONNECTION definition. This causes the user ID and password flowed with an ECI or EPI call to be authenticated with each request.

For the CICS TG on z/OS, specify `ATTACHSEC=IDENTIFY` in the CONNECTION definition to flow the user ID on an ECI call. The user ID and password is verified with RACF by the CICS TG before the ECI call is made. Any security failure on an ECI or EPI call will be returned as a generic security failure. Additionally, with the distributed CICS TG, you can use the ESI interface to authenticate a given user ID and password before making an ECI or EPI call.

Handling unregistered Secure Sockets Layer client certificates

To handle an unregistered SSL client certificate, the RACF certificate name filtering feature can be used to automatically assign a specific RACF user ID to a user according to predefined rules relating to the information contained in the certificate.

Mapping of many Web users to one Resource Access Control Facility user ID

To achieve this when using the CICS TG on z/OS, use SSL client certificates and the RACF certificate name filtering feature.

Restricting maximum permissions of a Web user

You might wish to prevent a user from running transactions from the Web because the user is authorized to run those transactions only in a non-Web environment. With CICS TG, you can achieve this by using a CICS link user ID, and restricting the authority to this user ID.

Handling expired passwords

If your users have no other means of managing their RACF user ID other than by using the CICS TG, you must consider how to give them the ability to change an expired password.

For CICS clients, the management of expired passwords can be handled by the ESI function CICS_ChangePassword and CICS_VerifyPassword. The ESI functions can be used only with CICS servers that support PEM. To use PEM, the Client daemon must be connected to the CICS server over TCP62 or SNA. An ESM such as RACF must also be available to the CICS server. ESI calls can be included within your ECI or EPI application.

When the CICS TG is on z/OS, use the PWAPI plug-in. A sample program is provided with the z/OS Web server that provides a function to identify when a user's password has expired, and a mechanism for changing the password.

Application access to authentication data

The CICS TG does not allow you to make decisions in your application based on authentication data. However, you can use the CICS TG security exits to access this information before your CICS application programs are called.

Sign-on support in CICS

For a distributed CICS TG using EPI, you can initiate a CICS transaction that uses the CICS CESN sign-on transaction or issues an EXEC CICS SIGNON command. Otherwise, when using the ECI, the user ID and password verified on the ECI call can be verified with RACF before the request is run in CICS.

6.5.2 CICS Transaction Gateway security checklist

The following list is a summary of the key actions that you must consider if you use the CICS TG to access your CICS applications. Note that most of these are normal security considerations, and you may decide that you do not have to implement all these actions. However, it is recommended that you consider each of them carefully.

- Use a firewall or a filtering router as the first line of defense. This allows you to restrict access to a specific set of IP addresses and ports, to filter out some potential Internet-style attacks, and to hide the actual IP address that your Web server or CICS TG is using.
- Use a means of authenticating all the users who access restricted information. You can do this by using SSL client certificates and using SSL to encrypt the user ID and password flows. You can also make use of link security and authenticate users coming in through the EPI.
- Do not transmit sensitive data as clear text across the Internet. Remember that this includes user IDs and passwords. Encryption of transmitted data is provided by SSL with or without client authentication.
- Prevent access to powerful administrative transactions such as CEMT or CECI if you are using the Terminal Servlet.
- Use a link user ID on the CICS connection definition to restrict access to your desired transactions and programs, regardless of the user ID flowed on an ECI or EPI call.
- Ensure that program autoinstall is turned off in CICS. Program autoinstall allows the loading into CICS storage of any program in the load libraries available to CICS. Without a program definition, a CICS program cannot run. Further, strictly control the programs in the load libraries available to the CICS region. If a program cannot be found, it cannot load and cannot run.

- Place limits on the maximum amount of system resources that can be used by transactions coming in from the CICS TG. Following are the ways to do this:
  - Set a maximum number of CICS TG connection manager threads in the CTG.ini file. This limits the maximum number of attached users.
  - Control the size of the CICS TG worker thread pool by setting a maximum number of worker threads in the CTG.ini file. This limits the maximum number of parallel ECI or EPI calls that can be issued by the CICS TG.
  - Use z/OS workload management restrictions on CPU usage for each address space such as the CICS TG Gateway daemon, the z/OS Web server, and CICS.
- Consider implementing resource security for transactions and other resources in your CICS region. This prevents malicious users from calling a program within your CICS region by using the CICS TG.
- Consider turning on transaction isolation and storage protection. This protects the CICS region from storage overlays from rogue applications accessing storage areas not associated with its active task.

7

WebSphere MQ

This chapter discusses the security requirements for connecting a CICS region to a WebSphere MQ subsystem. It discusses how to achieve secure communication using the Secure Sockets Layer (SSL) protocol and WebSphere MQ transport for SOAP.

This chapter discusses the following topics:

- CICS transaction security
- Access requirements for WebSphere MQ resources
- User ID authority for the CICS adapter
- The use of IBM Global Security Kit (GSKit) and the IBM Key Management (iKeyman) tool to configure SSL on WebSphere MQ channels
- The enablement of a configured SSL environment when using WebSphere MQ transport for SOAP

7.1 CICS transaction security

CICS transaction security can broken down into two requirements, the access requirements for long-running transactions and the access requirements for the user-initiated transactions.

7.1.1 Category One transaction definitions

WebSphere MQ has two transactions, CKTI and CKAM, which are designed to be run without a terminal. The CICS region user ID is required in order to have access to run these transactions. Example 7-1 shows how to define these transactions to the Resource Access Control Facility (RACF).

Example 7-1 Defining WebSphere MQ CICS Category One transactions to RACF

```
RALTER GCICSTRN CICSCAT1 ADDMEM(CKTI,CKAM)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

7.1.2 Administering the CICS adapter transactions

The following transactions are used to administer the WebSphere MQ CICS Adapter:

- CKQC: Controls the CICS adapter functions
- CKBM: Controls the CICS adapter functions
- CKRT: Controls the CICS adapter functions
- CKCN: Connect
- CKSD: Disconnect
- CKRS: Statistics
- CKDP: Full screen display
- CKDL: Line mode display
- CKSQ: CKTI Start/Stop
- CKMC: Distributed MQ (Channel Control)
- CKMH: Distributed MQ (Channel Help)
- CKRC: Distributed MQ (Receiver)
- CKRQ: Distributed MQ (Requestor)
- CKSG: Distributed MQ (Sender)
- CKSV: Distributed MQ (Server)

When these transactions are defined to CICS using the supplied samples, the RESEC option and the CMDSEC option are both set to NO. It is recommended that you do *not* change this setting. For more information, refer to *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454.

In order to provide a user with access to only display the current status of the adapter by using the full screen interface, they must be provided with access to transactions CKQC, CKBM, CKRT, and CKDP.

Example 7-2 shows how to define CICS WebSphere MQ transactions to RACF.

Example 7-2 Defining CICS WebSphere MQ transactions to RACF

```
RDEFINE GCICSTRN WSMQUSER UACC(NONE) +
ADDMEM(CKQC,CKBM,CKRT,CKCN,CKSD,CKRS,CKDP,CKDL,CKSQ,CKMC,CKMH,CKRC,CKRQ
,CKSG,CKSV)
PERMIT WSMQUSER CLASS(GICSTRN) ID(CICSUSER)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

7.2 Granting CICS access to WebSphere MQ

To enable the CICS region to connect to the WebSphere MQ subsystem, the CICS region must be authorized to connect to the WebSphere MQ subsystem.

The CICS region user ID or the group it is connected to must have READ access to RACF CLASS MQCONN profile <MQ sub system>.CICS. Example 7-3 shows how to grant access in order to enable CICS to connect to WebSphere MQ.

Example 7-3 Authorizing CICS to connect to WebSphere MQ

```
RDEFINE MQCONN MQMD.CICS
PERMIT MQMD.CICS CLASS(MQCONN) ID(CICS) ACCESS(READ)
SETROPTS RACLIST(MQCONN) REFRESH
```

7.3 Adapter user IDs

The user ID used by the CICS adapter is determined by the method in which the adapter interface is started. Irrespective of the startup process that is used, the user ID being used to start the CICS adapter must have access to all the WebSphere MQ resources.

Important: The user ID that is used to start the adapter is propagated to all the other transactions started by the CICS adapter, that is, when CKTI starts another CICS transaction as a result of a WebSphere MQ message arriving on a queue, it uses the user ID that was used to start the CICS adapter on the started transaction.

There are three different startup methods that can enable the CICS adapter (transaction CKTI) to have a different user ID:

- When the adapter is started during CICSPLT, including using the CICS System Initialization Table (SIT) option of MQCONN=YES, the CICS adapter task CKTI the user ID of the CKTI task is set to the user ID of the CICS region.
- When the CICS SIT option of PLTPIUSR is also set when the CICS adapter is started through the CICSPLT, the adapter executes under the user ID specified in the PLTPIUSR SIT option.

  Note: All the other tasks that start during the processing of the program list table (PLT) also start with the user ID specified in the SIT option PLTPIUSR.

- Starting the CICS Adapter after CICS initialization is completed. In this scenario, the CICS adapter can be started by using a sequential terminal, with the terminal having an input containing the CKQC STRATCKTI command or by using automation tools such as System Automation to sign in to CICS and issue the STARTCKTI command.

7.4 CICS initialization queue

WebSphere MQ uses an initialization queue to pass requests into CICS. When WebSphere MQ is started, the user ID starting the adapter interfaces is required in order to have UPDATE access to the CICS initialization queue. Example 7-4 shows granting CICS access to the initialization queue.

Example 7-4 Granting CICS access to the initialization queue

```
RDEFINE MQQUEUE CICS01.INITQ
PERMIT CICS.INITQ CLASS(MQQUEUE) ID(CICS) ACCESS(UPDATE)
SETROPTS RACLIST(MQQUEUE) REFRESH
```

7.5 Working with WebSphere MQ and SSL

This section discusses the use of SSL within a WebSphere MQ environment, including details about setting up secure message channels and secure Message Queue Interface (MQI) channels. The following steps are detailed in this section:

1. Creating the key repository
2. Generating the certificates for both the WebSphere MQ queue manager and the client
3. Adding digital certificates to the key repository, ready for use
4. Specifying the SSL attributes for the Universal Resource Indicator (URI) when using WebSphere MQ transport for SOAP

7.5.1 Configuring WebSphere MQ for secured communication

When using a message channel, it is vital that both the WebSphere MQ queue managers at each end have a user certificate. The channel is an MQI channel, and only if the SSLCAUTH attribute on the server connection channel is set to REQUIRED does the client on that channel also require a user certificate.

Note: When setting SSLCAUTH to OPTIONAL, if the client has a digital certificate, it is sent to the server and authenticated. If this authentication fails, the channel does not start. However, if the client does not have a certificate and SSLCAUTH is set to OPTIONAL, no attempt is made to authenticate, and the channel starts.

The tool that is used to demonstrate the key repository and certificate management is the GSKit because it is provided with the installation of WebSphere MQ. Some parts of this book assume that WebSphere MQ Explorer is already installed.

The SSL key repository

The SSL key repository is where all the digital certificates for use within WebSphere MQ are stored. To create an SSL key repository, perform the following tasks:

1. Open WebSphere MQ Explorer from **Start** → **Programs** → **IBM WebSphere MQ** → **WebSphere MQ Explorer**.
2. Start the GSKit key manager by right-clicking **IBM WebSphere MQ** and selecting **Manage SSL Certificates...** as shown in Figure 7-1.

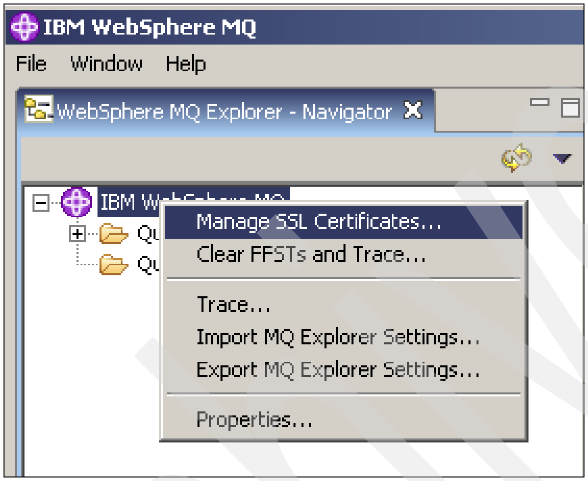
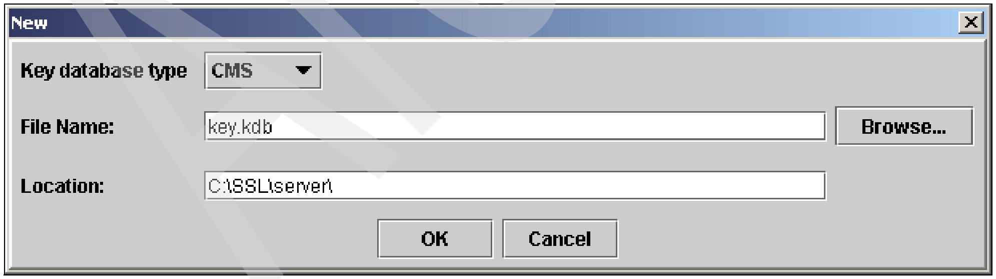

IBM WebSphere MQ
File
Window
Help
WebSphere MQ Explorer - Navigator
IBM WebSphere MQ
Qu
Qu
Manage SSL Certificates...
Clear FFSTs and Trace...
Trace...
Import MQ Explorer Settings...
Export MQ Explorer Settings...
Properties...

Figure 7-1 Opening the GSKit Key Manager

3. In the IBM Key Management tool, select **Key Database File** → **New....** Enter the File Name and Location to create the key repository, and click **OK**, as shown in Figure 7-2.



Key database type: CMS

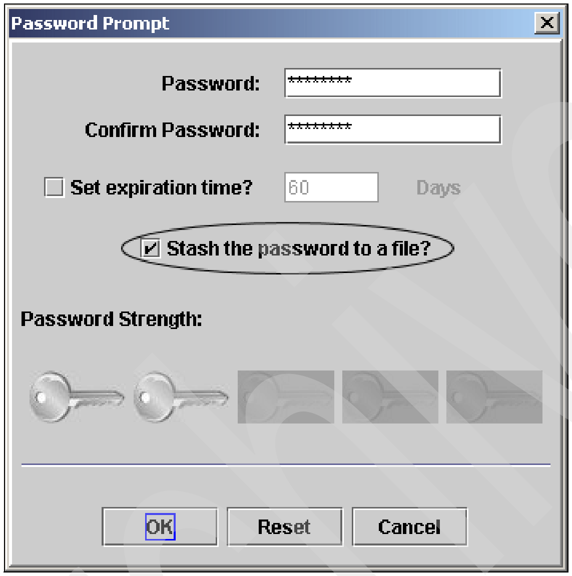
File Name: key.kdb

Location: C:\SSL\server\

Figure 7-2 Creating a new key repository

Note: If you are using the Java client, select **JKS** instead of the default CMS for the SSL key repository that is used by the client**.** This is due to SSL being handled by the Java Secure Socket Extension (JSSE). The file extension automatically changes to .jks.

4. Create a password to access the key repository, as shown in Figure 7-3. Select **Stash the password to a file?** and click **OK**.



Password Prompt

Password: *********

Confirm Password: *********

☐ Set expiration time? 60 Days

☑ Stash the password to a file?

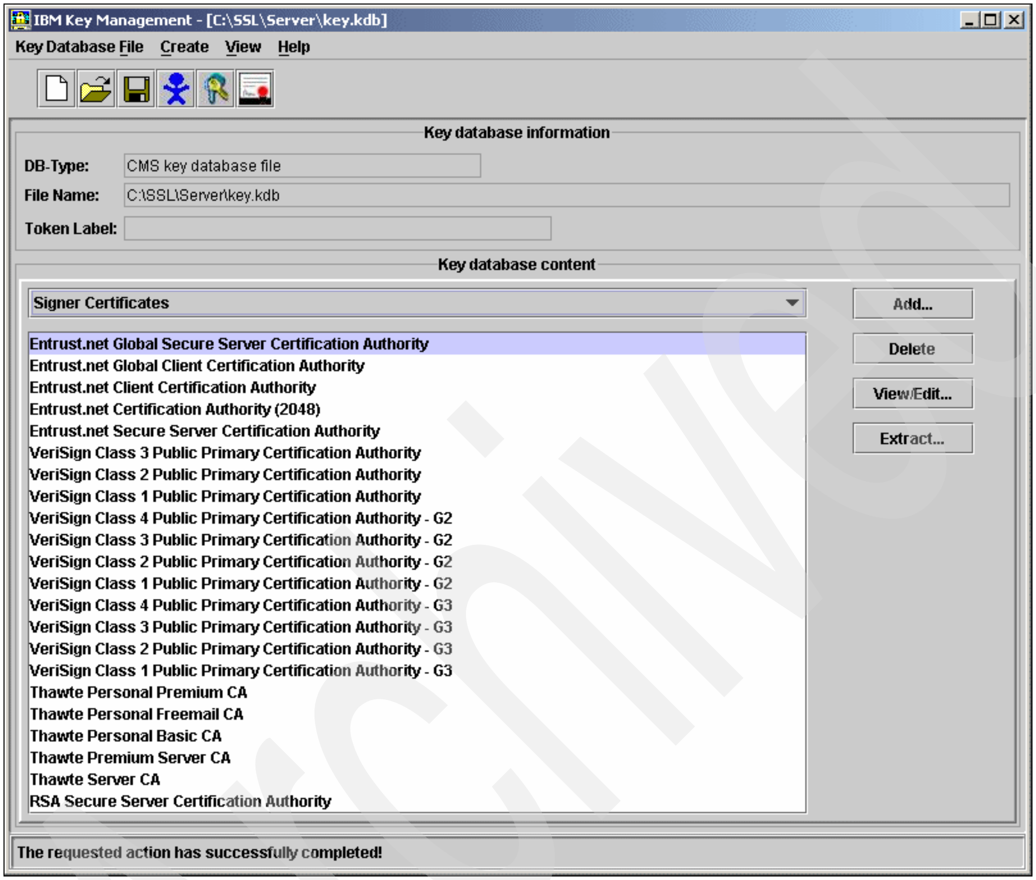
Password Strength:

Figure 7-3 Creating a password for the key repository

Important: It is vital that you select **Stash the password to a file?**. WebSphere MQ channels of any type does *not* start if the password is not stashed to a file. However, this is *not* required if you are using a Java client.

The password is stashed to a file in the same directory as the key repository, with the same file name, but with an .sth extension.

The key repository is created. By default, when you first create a key repository, it contains a selection of trusted root certificate authority (CA) certificates. This is shown in Figure 7-4.



DB-Type: CMS key database file

File Name: C:\SSL\Server\key.kdb

Token Label:

Signer Certificates

Entrust.net Global Secure Server Certification Authority
Entrust.net Global Client Certification Authority
Entrust.net Client Certification Authority
Entrust.net Certification Authority (2048)
Entrust.net Secure Server Certification Authority
VeriSign Class 3 Public Primary Certification Authority
VeriSign Class 2 Public Primary Certification Authority
VeriSign Class 1 Public Primary Certification Authority
VeriSign Class 4 Public Primary Certification Authority - G2
VeriSign Class 3 Public Primary Certification Authority - G2
VeriSign Class 2 Public Primary Certification Authority - G2
VeriSign Class 1 Public Primary Certification Authority - G2
VeriSign Class 4 Public Primary Certification Authority - G3
VeriSign Class 3 Public Primary Certification Authority - G3
VeriSign Class 2 Public Primary Certification Authority - G3
VeriSign Class 1 Public Primary Certification Authority - G3
Thawte Personal Premium CA
Thawte Personal Freemail CA
Thawte Personal Basic CA
Thawte Premium Server CA
Thawte Server CA
RSA Secure Server Certification Authority

The requested action has successfully completed!

Figure 7-4 The default CA root certificates in a new key repository

The graphical view of the key repository provides you with the ability to look at the CA (or signer) certificates, the user or personal certificates, and any certificate requests for personal certificates that have been generated. To access each of these, click the dropdown box named **Signer Certificates**. Each of the different views provides different action buttons along the right side of the window. For more information about how to use these, refer to *IBM Tivoli Access Manager Secure Sockets Layer Introduction and iKeyman Users Guide, V5.1*, SC32-1363.

For purposes of demonstration, the remaining tasks in certificate management is performed using the command-line version of the iKeyman graphical user interface (GUI). Before using the command-line tool, execute the following commands to ensure that the environment is correctly configured, as shown in Example 7-5.

Example 7-5 Ensuring that the environment is correctly configured

```
set PATH=%PATH%;C:\Program Files\IBM\gsk7\bin\
set JAVA_HOME=C:\Program Files\IBM\WebSphere MQ\gskit\jre\
```

These are the default install directories for the command-line version of iKeyman and the Java installation used by iKeyman.

On UNIX platforms, the JAVA_HOME variable must be set as specified in Table 7-1.

Table 7-1 Environment setup for UNIX platforms

Platform	**Command**
AIX	export JAVA_HOME=/usr/mqm/ssl/jre
HP-UX	export JAVA_HOME=/opt/mqm/ssl
Linux	export JAVA_HOME=/opt/mqm/ssl/jre
Solaris	export JAVA_HOME=/opt/mqm/ssl

The Java SSL key repository

When using the Java client, some extra options must be specified for authentication to succeed during communication. The JSSE handles the SSL functionality and prescribes that there must be a *trust store* and a *key store*. Each of these are represented as a standard SSL key store, as described in the previous section. However, it is their contents that define whether they are a trust store or a key store:

- A trust store contains only CA certificates and can be used by every user on a given system as a reference to the trusted authorities
- A key store contains at least a personal (or user) certificate. Optionally, it may also contain a list of CA certificates. It must be specific to one user on the system.

It is possible for a key repository to behave as both a trust store and a key store at the same time. The contents of a key store can only be accessed using the correct password. The CA certificate contents of the key repository can be viewed by anyone without a password. Therefore, a key store can be used as a trust store too and still preserve the confidentiality of the personal certificate. When the key repository is used in this manner, it can be treated the same way as the standard key repository, as described earlier.

To use a separate key repository for each trust store and key store, the process of creating a key store (with a .jks extension) must be followed twice, once for each type. Care must be taken when generating and importing a CA certificate, and subsequently, when signing and adding user certificates to the correct repositories. In the commands that follow, note the differences in the option and parameters when using the Java client.

Creating the root CA certificate using the GSKit

In the following examples, each of the user certificates have been signed by a CA certificate. This section details how to generate the CA or the signer certificate. The iKeyman provides the facility to create a user certificate that is self-signed. This means that the certificate is a root CA certificate, which can also be used as a user certificate. In the steps that follow, the self-signed certificate is created as a normal user certificate, and then moved around the key repository so that it becomes a true CA or signer certificate, which is then used to sign more user certificates. These instructions assume that a temporary key repository is already created, holds the generated CA certificate, and is used to sign certificate requests.

1. Execute the command shown in Example 7-6 to create a self-signed certificate and add it to a temporary key repository.

Example 7-6 Creating a self-signed certificate and adding it to a temporary key repository

```
gsk7cmd -cert -create -db "C:\SSL\temp\key.kdb" -pw password -label
"Root CA Certificate" -dn "CN=Root CA, O=IBM, OU=ITSO, C=US" -expire
1000
```

> **Note:** When using the Java client, the db value has the extension .jks instead of .kdb.

The values for *db* and *pw* must be the location of the key repository and the password to access it, respectively. *Label*, *dn*, and *expire* can be set to anything suitable, according to the system requirements. (The value of *expire* is the number of days until this certificate becomes invalid.)

2. Extract the certificate from its current key repository, so that it can be used as a root CA or signer certificate for use in the demonstrations that follow. To extract the certificate, execute the command shown in Example 7-7.

Example 7-7 Extracting the certificate

```
gsk7cmd -cert -extract -db “C:\SSL\temp\key.kdb” -pw password -label
“Root CA Certificate” -target “C:\SSL\temp\CAExtracted.arm” -format
ascii
```

> **Note:** When using the Java client, the db value has the extension .jks instead of .kdb.

   This command creates a file called CAExtracted.arm in the same directory as the key repository. If you are using the American Standard Code for Information Interchange (ASCII) format, the file extension must be .arm. If you are using the binary format, the file extension must be .der. These are the file extensions prescribed by the GSKit and the iKeyman tool.

3. Retain the key repository used to create the CA certificate in order to sign the certificate requests.

Securing the channels between the queue managers

Secure the message channels between two server WebSphere MQ queue managers using SSL by performing the following tasks:

1. Create an SSL key repository for each queue manager, ensuring that the password for each repository is stashed. For more information about the tasks you must perform, refer to “The SSL key repository” on page 222.
2. Each queue manager’s key repository requires the root CA certificate in its signer certificate store. Issue the command shown in Example 7-8 for each queue manager key repository. In this command, the value of the file must be the location of the file that was extracted from the temporary key repository, in this case, CAExtracted.arm.

Example 7-8 Command to create CA certificate

```
gsk7cmd -cert -add -db “C:\SSL\server\key.kdb” -pw password -label
“Root CA Certificate” -file “C:\SSL\temp\CAExtracted.arm” -format ascii
-trust enable
```

3. Each queue manager requires a user certificate. Therefore, a certificate request must be constructed. Issue the command shown in Example 7-9 for each queue manager.

> **Important:** Prefix the label option in this command with `ibmwebspheremq` followed immediately by the name of the queue manager folded to lower case. In Example 7-9, the queue manager name is QM1, and therefore, the label for the certificate is ibmwebspheremqqm1.

Example 7-9 Constructing a user certificate

```
gsk7cmd -certreq -create -db “C:\SSL\server\key.kdb” -pw password
-label “ibmwebspheremqqm1” -dn “CN=QM1, O=IBM, OU=ITSO, C=US” -file
“C:\SSL\server\ibmwebspheremqqm1_request.arm”
```

4. At this point, the certificate requests generated in step 3 are sent to an external CA. However, this example has its own CA, and can therefore take these certificate requests and process them locally. Issue the command shown in Example 7-10 for each certificate request. The signed certificate is stored in the file ibmwebspheremqqm1_sigend.arm.

> **Note:** In the command shown in Example 7-10, the value of db is the key repository in which the original CA certificate was created. The original self-signed CA certificate holds both the public key and the private key used to sign any certificate requests.

Example 7-10 Command for each certificate request

```
gsk7cmd -cert -sign -file “C:\SSL\server\ibmwebspheremqqm1_request.arm”
-db “C:\SSL\temp\key.kdb” -pw password -label “Root CA Certificate”
-target “C:\SSL\server\ibmwebspheremqqm1_sigend.arm” -expire 364
```

5. You can add the user certificates to each queue manager’s key repository, for example, for a queue manager called QM1, issue the command shown in Example 7-11.

Example 7-11 Adding user certificates to each queue manager’s key repository

```
gsk7cmd -cert -receive -db “C:\SSL\server\key.kdb” -pw password -file
“C:\SSL\server\ibmwebspheremqqm1_signed.arm”
```

6. Issue a similar command for the other server queue manager in order to get its user certificate into its own key repository. At this point, it is necessary to make sure that the queue manager attribute SSLKEYR is set correctly for each queue manager. Example 7-12 shows how to set the SSKLEYR value to the right location.

   Example 7-12 Altering the SSLKEYR value for a queue manager

   ```
   >runmqsc QM1
   5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
   Starting MQSC for queue manager QM1.

   alter qmgr SSLKEYR(‘C:\SSL\server\key’)
      1 : alter qmgr SSLKEYR('C:\SSL\server\key')
   AMQ8005: WebSphere MQ queue manager changed.
   ```

   This step is necessary, unless the default location for the key repositories has been used, and the default name of the repository file has *not* been changed from key.kdb. When altering the value of SSLKEYR, the .kdb extension must *not* be used, and the secured SSL channels must not be started.

7. To enable SSL on channels, it is necessary to set the SSLCIPH value on the channel to a valid cipherSpec. The same cipherSpec must be specified at both ends of the channel. Example 7-13 shows one such instance.

   Example 7-13 Altering the SSLCIPH value on a channel

   ```
   >runmqsc QM1
   5724-H72 (C) Copyright IBM Corp. 1994, 2004.  ALL RIGHTS RESERVED.
   Starting MQSC for queue manager QM1.

   alter channel(QM1_2_QM2_SDRC) chltype(SDR) SSLCIPH(RC2_MD5_EXPORT)
       4 : alter channel(QM1_2_QM2_SDRC) chltype(SDR)
   SSLCIPH(RC2_MD5_EXPORT)
   AMQ8016: WebSphere MQ channel changed.
   ```

For more information about the available cipherSpec, refer to *WebSphere MQ Security,* SC34-6588. Changes to the SSL settings on channels are only picked up when the channels are restarted. At this point, when you restart the channels, the SSL becomes enabled and active.

The commands shown here are suitable for both the Windows platform and the UNIX platform. The examples shown here are for the Windows systems. The only alteration required for UNIX platforms is to change the way the directory structure is passed to the **gsk7cmd** command.

Securing the client/server connections

Enabling SSL on an MQI channel is slightly different from enabling SSL on a message channel. However, many of the same ideas apply, and many of the commands described earlier are valid. This section highlights the differences between the two types of channels, and describes the process involved in enabling SSL between a WebSphere MQ client and a server queue manager.

WebSphere MQ clients make use of both an SSL key repository (in the case of the Java client, this is a Java key store) and a user certificate that is signed by a CA. Therefore, the setup stages for an MQI channel are almost the same. The first difference occurs in 3 on page 228 described in the previous section. The label for a client certificate must conform to following syntax:

```
ibmwebspheremqlogonid
```

In the example described in the previous section, for a message channel, the suffix is the queue manager name folded to lower case. However, for a client certificate, the suffix must be the login ID for the client system user. Therefore, issue the command shown in Example 7-14 to create a certificate request for a client.

> **Note:** If you are using a Java client, the file extension for the db value must be .jks. This relates directly to the type of key store created for the client. Moreover, the value of the db must be the Java key store that holds the user certificate.

Example 7-14 Command to create a certificate for a client

```
gsk7cmd -certreq -create -db “C:\SSL\client\key.kdb” -pw password
-label “ibmwebspheremqmmyuserid” -dn “CN=Client 1, O=IBM, OU=ITSO,
C=US” -file “C:\SSL\client\ibmwebspheremqmyuserid_request.arm”
```

The certificate request is then signed in the usual manner for all client types, with the command shown in Example 7-15.

> **Note:** As with the command shown in Example 7-14, here too, the file extension for the db value must be .jks if you are using the Java client.

Example 7-15 Signing the certificate request

```
gsk7cmd -cert -sign -file
“C:\SSL\client\ibmwebspheremqmyuserid_request.arm” -db
“C:\SSL\temp\key.kdb” -pw password -label “Root CA Certificate” -target
“C:\SSL\client\ibmwebspheremqmyuserid_sigend.arm” -expire 364
```

Add the signed certificate as a user certificate in the client’s key repository by issuing the command shown in Example 7-16.

> **Note:** As with the command shown in Example 7-14, here too, the file extension for the db value must be .jks if you are using the Java client.

Example 7-16 Adding the signed certificate as a user certificate

```
gsk7cmd -cert -receive -db “C:\SSL\client\key.kdb” -pw password -file
“C:\SSL\client\ibmwebspheremqmyuserid_signed.arm”
```

If the system requires a separate Java trust store, add the CA certificate that has signed the user certificate to that trust store. To do this, issue the command shown in Example 7-17.

> **Important:** This step is *not* required if the key store is also being used as the trust store.

Example 7-17 Adding a CA certificate to the trust store

```
gsk7cmd -cert -add -db “C:\SSL\Java\client\trust.jks” -pw password
-label “Root CA Certificate” -file “C:\SSL\temp\CAExtracted.arm”
-format ascii -trust enable
```

To inform a client about the location of its SSL key repository, use of one the following options:

- The simplest way is to define an environment variable, MQSSLKEYR. Table 7-2 shows how to set this up on the Windows platform and the UNIX platform.

  Table 7-2 Setting the MQSSLKEYR environment variable

Platform	Command (example)
Windows	set MQSSLKEYR=C:\client\key
UNIX	export MQSSLKEYR=/client/key

  As mentioned earlier, the .kdb extension is *not* required, and adding it to the end of the file name means that the client is unable to find the key repository.

- Alternatively, the location of the SSL key repository can be specified as part of the SSL connect options structure (MQSCO) that is used when issuing a MQCONNX call from with an application.

It then becomes the application’s responsibility to update the KeyRepository field within the MQSCO to the location of the SSL key repository, again without the .kdb extension. For more information about the MQSCO, refer to *WebSphere MQ Application Programming Reference,* SC34-6596.

Specifying the location of the Java trust store and the Java key store to the Java client is a little more complex. It is performed using the -d option in the **`java`** command. Set the following three parameters:

- javax.net.ssl.keyStore
- javax.net.ssl.keyStorePassword
- javax.net.ssl.trustStore

The command shown in Example 7-18 executes the client application and sets the specified parameters.

Example 7-18 Executing the client application and setting the specified parameters

```
java -Djavax.net.ssl.keyStore=C:\SSL\Java\client\key.jks
-Djavax.net.ssl.keyStorePassword=password
-Djavax.net.ssl.trustStore=C:\SSL\Java\client\trust.jks clientApp
```

In this example, clientApp is the compiled Java code, representing the client application.

The cipherSpec can be specified in three ways. The client connection channel has a SSLCIPH attribute that is used to provide the type of algorithm used for authenticating and encrypting the data flowing across the channel. Although it is possible to enable each of the three methods, the following list shows the methods in the order of precedence:

- Within an MQCONNX call

  Within the channel definition structure MQCD, there is a field named SSLCipherSpec, which you can use to specify the cipherSpec that is to be used for communication.

- Using a client channel definition table

  If the client system is unable to access the client channel definition table as a shared file on the server machine, the definition copies the table to the client machine. As an entry in the definition table, the SSLCIPH is defined and altered on the server machine before being copied to the client machine. For more information about using the client channel definition tables and the setup that is required, refer to *WebSphere MQ Clients,* GC34-6590.

- Using Active Directory® on Windows

  On Windows systems, it is possible to publish the client channel definition tale to Active Directory by using the **setmqscp** command. For more information about using Active Directory, refer to *WebSphere MQ System Administration Guide,* SC34-6584.

When you use a Java client, the equivalent value of SSLCIPH is SSLCIPHERSUITE. Configure it by using the JMSAdmin tool. For more information about specifying SSL on a Java client, refer to *WebSphere MQ Using Java,* SC34-6591.

Specify the location of the key repositories only when a WebSphere MQ client or a Java client is communicating with a server queue manager in the usual manner. Within the area of Web services, the SSL configuration options are specified using the Universal Resource Indicator (URI).

SSL in the Universal Resource Indicator

The SSL configuration for WebSphere MQ transport for SOAP is provided in the form of several options in the WebSphere MQ URI. The options that are available depend on the environment being used.

In a Microsoft .NET environment, the following options are specific to the environment:

- sslKeyRepository

  This is the location of the SSL key repository, specified as a full path name to the .kdb file. Specify this *without* the .kdb extension.

- sslCipherSpec

  This can be any value that can be specified in the SSLCIPH value on any channel. For a full list of the possible values, refer to *WebSphere MQ Security,* SC34-6588. This is a mandatory field if sslKeyRepository is set.

In a Java environment, the following options are specific to the environment:

- sslKeyStore

  This is the location of the JSSE key store, specified as a full path name to the .jks file. Specify this *without* the .jks extension.

- sslKeyStorePassword

  This is the password that grants access to the JSSE key store specified in the sslKeyStore option.

- sslTrustStore

  This is the location of the JSSE trust store, specified as a full path name to the .jks file. Specify this *without* the .jks extension.

- sslTrustStorePassword

  This is the password that grants access to the JSSE trust store specified in the sslTrustStore option.

- sslCipherSuite

  This is the cipherSuite as specified in *WebSphere MQ Using Java,* SC34-6591. In a Java environment, there is a direct mapping from the value of a cipherSuite to cipherSpec, as used in the message channels.

If any of these are specified in an environment that they are not specific to, they are ignored. For more information about the SSL options available on the WebSphere MQ URI, refer to *WebSphere MQ Transport for SOAP,* SC34-6651.

Another important, but optional option that can be set on the URI is SSLPeerName. This represents a section of the distinguished name (DN) that must be in any certificate received from the remote participant in the communication. (This is the certificate the WebSphere MQ queue manager receives from the client). The value of SSLPeerName can contain the asterisk (*) character, representing a wild card, for example, an SSLPeerName of CN=IBM* matches CN=IBM Corporation.

Chapters 8 - 13 of this book discuss the implementation of Web services and clients to invoke them within a .NET, Axis, and IBM WebSphere Application Server environment. Each client uses WebSphere MQ transport for SOAP to invoke the Web service. Therefore, SSL support within WebSphere MQ is integral to securing the communication from the client through WebSphere MQ on to invocation of the Web service, and the response that flows back.

Within the scope of this book, there are three distinct areas in which securing communication using SSL is important:

- The connections between the server queue managers within the WebSphere MQ network topology
- The way the invoking client connects to a queue manager
- The way the invoked Web service connects to a queue manager

For more details about how to secure communication between the server queue managers, refer to *WebSphere MQ Security*, SC34-6588.

The subsequent chapters focus on securing the client/server connections between the invoking client and a queue manager, and the invoked Web service and a queue manager. The enablement of SSL comes entirely from the settings of the options supplied in the URI. The underlying implementation of the security services that SSL supplies is achieved by putting in place the key repositories and populating each of them with the required certificates. This chapter provided details about the tasks involved in implementing these security services. Each of the subsequent chapters detail the enablement of the services provided by WebSphere MQ and SOAP.

8

CICS Enterprise JavaBeans support

Accessing enterprise beans requires the use of the Internet Inter-ORB Protocol (IIOP). Support for IIOP is provided by the CICS TCP/IP listener. Figure 8-1 shows a typical CICS enterprise bean environment.

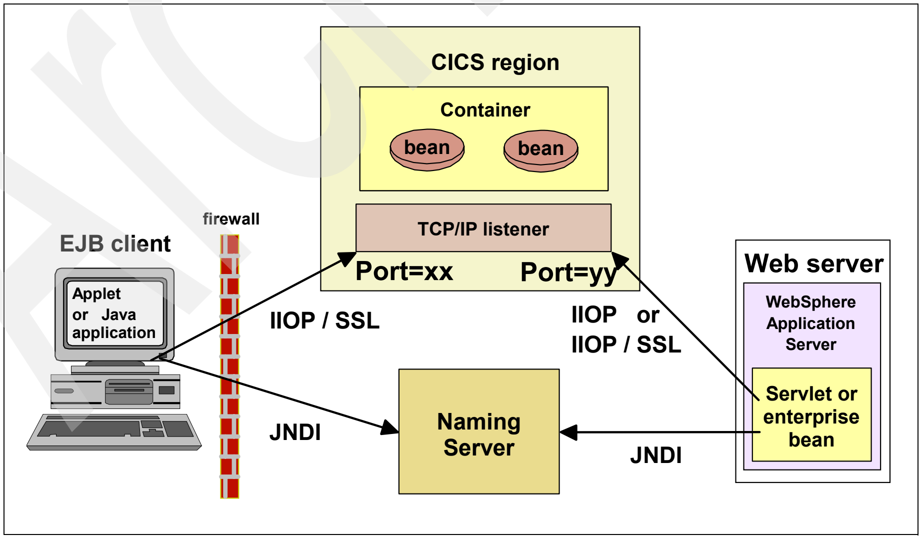


Figure 8-1 Using SSL with EJB clients

Applets or Java applications may require the use of Secure Sockets Layer (SSL). But a servlet or an enterprise bean running on IBM WebSphere may not require SSL security because CICS and WebSphere are likely to be in your private network.

Thus, providing security for enterprise beans involves:

- Providing an appropriate level of privacy and data integrity (SSL support)
- Confirming the identity of the client (authentication)
- Controlling what the client can do (authorization)

8.1 Secure Sockets Layer support

To activate SSL support for an *incoming* IIOP request for an enterprise bean, specify one of the following for the value of the SSL parameter of the TCPIPSERVICE definition:

- YES
- CLIENTAUTH

CICS Transaction Server (TS) V2.2 and V2.3 also support *outbound* IIOP requests. This enables an enterprise bean in a CICS Enterprise JavaBeans (EJB) Server to invoke an enterprise bean in another EJB Server using Remote Method Invocation over IIOP (RMI/IIOP). This communication can also be encrypted using SSL. The Interoperable Object Reference (IOR) of the remote enterprise bean specifies whether SSL must be used.

The remote server can request CICS to authenticate itself with a client certificate. CICS finds the label of the client certificate that it must use in the CERTIFICATE parameter of the CORBASERVER definition and then obtains the certificate from the KEYRING, as specified in the System Initialization Table (SIT).

8.1.1 Improved Secure Sockets Layer support in CICS TS V2.3

This section discusses how CICS TS V2.3 allows you to restrict the cipher suites that CICS advertises for connections that use the Hypertext Transfer Protocol (HTTP). Use the same values of the PRIVACY parameter of the TCPIPSERVICE definition, namely:

- NOT SUPPORTED
- SUPPORTED
- REQUIRED

This is to also restrict the cipher suites that CICS advertises for *inbound IIOP* requests.

To restrict the cipher suites that CICS uses for *outbound IIOP* requests, specify one of the same values for the OUTPRIVACY parameter of the CORBASERVER definition.

8.2 Authentication

Because CICS authorization is based on a user ID, CICS must derive one from the IIOP request and authenticate it. It can do so by using the information associated with or provided by the following:

- An SSL client certificate
- The security user-replaceable module (DFHXOPUS)
- Asserted identity

Secure Sockets Layer client certificate

If you are using SSL to send an IIOP message, use client certificates to authenticate the user. This is achieved by setting SSL(CLIENTAUTH) and AUTHENTICATE(CERTIFICATE) in the TCPIPSERVICE definition and CLIENTCERT(tcpipservicename) in the CORBASERVER definition.

The mapping from a certificate to a user ID exists when:

- The certificate is already registered to a user ID in your external security manager's database (single certificate)
- The information sent in the client certificate matches a Resource Access Control Facility (RACF) Certificate Name Filtering profile that allows multiple SSL client certificates to be associated with a single user ID.

Security user-replaceable module (DFHXOPUS)

If no SSL client certificate is provided, assign a user ID by coding a user-replaceable module (URM) and specifying its name in the URM parameter of the TCPIPSERVICE definition. If you do not specify a module name, CICS assigns the default user ID set in the DFLTUSER system initialization parameter.

CICS provides the sample DFHXOPUS module, which is a C program. An alternative COBOL version (named COBXOPUS) is available as additional material for the IBM Redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284.

Because it is possible for a CICS region to have several TCP/IP listeners, you can use different URMs. The URM must return a pointer to an 8-byte character field in its COMMAREA, and CICS will associate the user ID in that field with the server transaction.

The URM can use a variety of designs to choose the correct identification. Following is a list of the possibilities:

- Passing a user ID and password in the IIOP message and performing custom authentication using the EXEC CICS VERIFY PASSWORD command
- Extracting user information such as the client TCP/IP address and port number, using the EXEC CICS EXTRACT TCPIP command
- Mapping user IDs based on enterprise bean names and method names
- Assigning a single user ID for all IIOP messages

Asserted identity

Assume that an HTTP request drives a servlet in WebSphere, which then invokes a method on an enterprise bean in CICS, as shown in Figure 8-2.

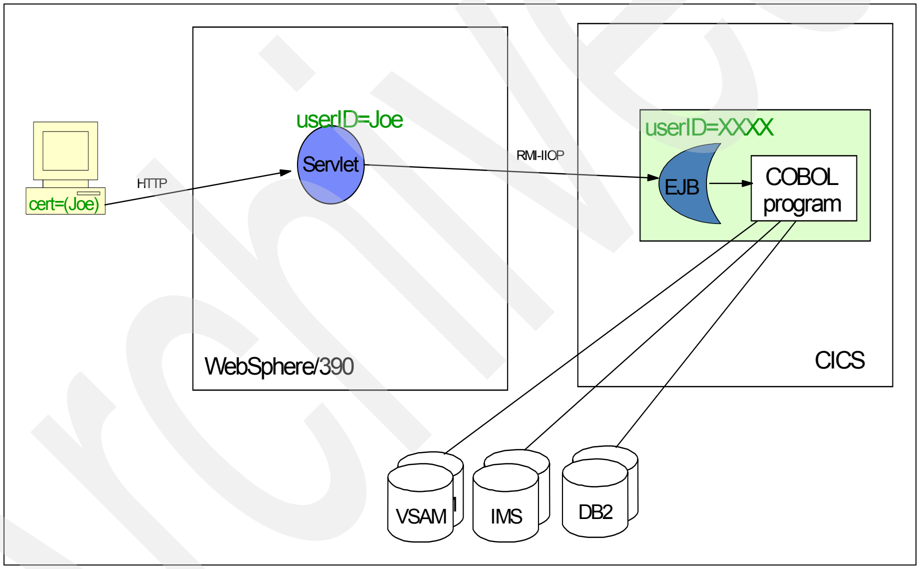


Figure 8-2 Invoking a bean method through an intermediate server

In CICS TS V2.2, CICS does *not* recognize that there is a server at the other end of the IIOP connection. WebSphere may present a client certificate, but it is not the user's certificate. Rather, it is WebSphere's own certificate, and thus WebSphere's identity. Thus, the work in CICS executes under some user ID other than "Joe". This may be the CICS region default user ID, one assigned by the URM, or the one associated with a certificate presented by WebSphere.

IBM has developed a proprietary protocol called Asserted Identity to address this problem. Implemented by WebSphere V4 and WebSphere V5 and CICS TS V2.3, Asserted Identity allows “Joe” to be known in all the servers and his security identification will be passed from WebSphere to CICS.

Asserted identity authentication can be used when an IIOP client communicates with the target server (CICS TS V2.3) through an intermediate server (CICS TS V2.3, WebSphere V4 or WebSphere V5), and both the servers use the same security manager. It works as follows:

1. The intermediate server’s identity is authenticated by the target server using SSL client certificate authentication.
2. Through the security manager, the target server verifies that the intermediate server can be trusted to authenticate its clients.
3. When the intermediate server receives a request, it authenticates the client using whatever authentication protocol is appropriate. If the client is successfully authenticated, the intermediate server passes the request to the target server.
4. Because the target server trusts the intermediate server to authenticate the client, it makes no further checks of the client’s authenticity before processing the client’s request.

To establish a trust relationship between the intermediate and target servers, where the target server is a CICS CorbaServer, perform the following tasks:

1. Configure your CICS region to use SSL authentication and specify `AUTHENTICATE(ASSERTED)` and `SSL(CLIENTAUTH)` in the TCPIPSERVICE definition.
2. Associate the intermediate server’s client certificate with a RACF user ID.
3. Create a profile named DFH.*applid.corbaserver*.ASSERTID in the SERVAUTH general resource class, where *applid* is the APPLID of the CICS region and *corbaserver* is the name of the target CorbaServer. Use the following RACF command:

   ```
   RDEFINE SERVAUTH DFH.applid.corbaserver.ASSERTID UACC(NONE)
   ```

4. Provide the intermediate server’s user ID (established in step 2) READ authority to the profile. Use the following RACF command, for example:

   ```
   PERMIT DFH.applid.corbaserver.ASSERTID CLASS(SERVAUTH)
   ID(server_user_ID) ACCESS(READ)
   ```

5. If the intermediate server is WebSphere, specify `Send asserted identities allowed` in its Properties form. Fill in the values for the SSL-related elements (SSL RACF-keyring, SSL V2 timeout, and SSL V3 timeout).

If the intermediate server is CICS, no additional setup is required. CICS always sends asserted identities if the target server is capable of receiving them.

User ID authentication summary

Table 8-1 summarizes the relation between the TCPIPSERVICE and the CORBASERVER parameters relating to SSL, and shows how the user ID is obtained.

Table 8-1 User ID authentication summary

Authentication method	**TCPIPSERVICE AUTHENTICATE parameter**	**TCPIPSERVICE SSL parameter**	**Associated CORBASERVER parameter**	**Client Cert associated with user ID**	**How the user ID of the IIOP client is identified**
IIOP with no authentication	NO	NO	UNAUTH (*tcpipservice*)	N/A	User ID provided by URM specified on TCPIPSERVICE(2)
IIOP with no authentication	NO	YES	SSLUNAUTH (*tcpipservice*)	N/A	User ID provided by URM specified on TCPIPSERVICE (2)
IIOP with SSL client certificate optional	NO	CLIENTAUTH	SSLUNAUTH (*tcpipservice*)	NO	User ID provided by URM specified on TCPIPSERVICE(2)
				YES	That user ID is used
IIOP with SSL client certificate required	CERTIFICATE	CLIENTAUTH	CLIENTCERT (*tcpipservice*)	NO	Connection is rejected
				YES	That user ID is used

Authentication method	TCPIPSERVICE AUTHENTICATE parameter	TCPIPSERVICE SSL parameter	Associated CORBASERVER parameter	Client Cert associated with user ID	How the user ID of the IIOP client is identified
IIOP with asserted identity authentication	ASSERTED	CLIENTAUTH	ASSERTED (*tcpipservice*)	NO (1)	Connection is rejected
				YES (1)	User ID sent in IIOP request by intermediate server

1. Intermediate server’s certificate
2. If a program is not specified in the URM parameter of the TCPIPSERVICE definition, the CICS default user ID is used

8.3 Authorization

Authorization involves verifying that a user is allowed to access a particular method in a bean, and then to access the resources used by that method.

Associating methods with transaction IDs

CICS runs each method call as a CICS transaction. The association between a method and a CICS transaction is specified in a REQUESTMODEL resource definition. A CICS-supplied transaction, CREA, enables you to generate REQUESTMODEL definitions that can be installed dynamically into the CICS region, written to the CICS system definition file (CSD), or both. CREA displays a list of beans and bean methods within the JAR file related to an installed DJAR resource definition, and enables you to associate transaction IDs with those beans and methods. After you have finished associating transaction IDs with beans and methods, a list of request models that best match the transaction IDs with the beans and methods in the JAR file, is generated. Each REQUESTMODEL definition is presented to you with the option to:

- Install the REQUESTMODEL in CICS
- Define the REQUESTMODEL to the CSD
- Install and define the REQUESTMODEL
- Ignore the REQUESTMODEL

A list of the REQUESTMODELs that you have installed and defined is then displayed. Any REQUESTMODELs installed in CICS that are *not* used by the DJAR, but refer to beans within the DJAR, are also shown. Similarly, any transaction IDs that are used, but currently not installed, are also listed.

Matching method calls to REQUESTMODELs

Figure 8-3 illustrates the process of matching a method request to a REQUESTMODEL. CICS uses the bean name, the interface type (home or remote), the CorbaServer name, and the method name (in this order) to match the incoming method request with a REQUESTMODEL. A more specific match overrides a generic match. If you do not define a REQUESTMODEL, CICS uses the default REQUESTMODEL, which always maps the method name to transaction CIRP. The CIRP transaction points to the default request processor DFJIIRP.

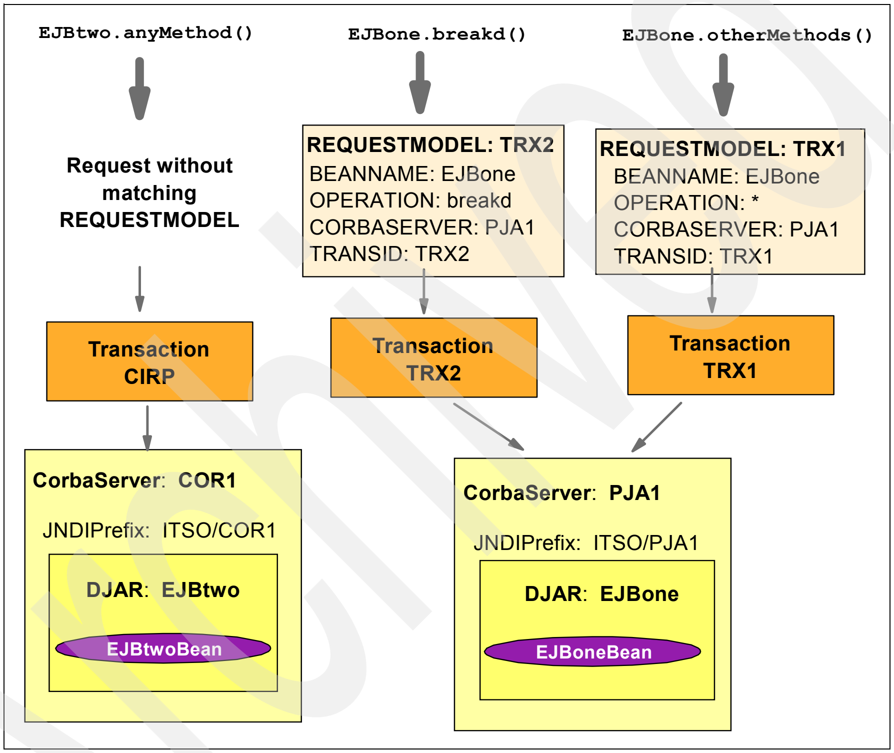


Figure 8-3 Using REQUESTMODELs to map a bean method to a transaction ID

In your external security manager, provide access to the transaction CIRP to the user ID associated with any request for a method in EJBtwo. Similarly, provide access to transaction TRX2 to the user ID associated with a client request for the breakd() method in EJBone, and access to transaction TRX1 to the user ID associated with a client request for any other method in EJBone.

8.4 Security roles

In the Enterprise JavaBeans Specification V1.1, access to enterprise bean methods is based on the concept of security roles. A security role represents a type of user of an application in terms of the permissions that user must have to successfully use the application. A role is a logical security identification.

In a payroll application, for example:

- A manager role could represent users who are permitted to use all parts of an application
- A team_leader role could represent users who are permitted to use the administrative functions of the application
- A data_entry role could represent users who are permitted to use the data entry functions of the application

The security roles for an application are defined by the application assembler, and are specified in the bean's deployment descriptor, as shown in Figure 8-4.

```
<security-role>
   <description>
       A user with this role may access any method
   </description>
   <role-name>
       Manager
   </role-name>
</security-role>
<security-role>
   <description>
       A user with this role may use administrative functions
   </description>
   <role-name>
       Team_leader
   </role-name>
</security-role>
```

Figure 8-4 Definition of security roles in an XML deployment descriptor

The security roles that are permitted to execute a bean method are also specified in the bean's deployment descriptor, again by the application assembler. The association between security roles and the bean methods is defined by using method-permission tags.

Figure 8-5 shows an example of providing the security role Manager with access to all the methods in the bean named Payroll.

```
<method-permission>
   <role-name>Manager</role-name>
      <method>
         <ejb-name>Payroll</ejb-name>
            <method-name>*</method-name>
      </method>
</method-permission>
```

Figure 8-5   Access rights for the Manager role

In this example, methods that update the number of hours worked by employees each week may be assigned to the data_entry role, and methods that delete an employee from the payroll may be assigned to the team_leader role.

To distinguish similarly-named security roles in different applications or on different systems, the security roles specified in the bean's deployment descriptor can be given a one-part or two-part qualifier when the bean is deployed in a CICS system:

- Security role with no qualifier: Team_leader
- Security role with one qualifier: Payroll.team_leader
- Security role with two qualifiers: Prodcics.payroll.team_leader

In this example, `payroll` qualifies the security role at the application level and is used to distinguish between the team_leader role in the payroll application and the team_leader role in the other applications. Similarly, prodcics qualifies the security role at the system level and is used to distinguish between the payroll.team_leader role in the prodcics region and the payroll.team_leader role in the test regions.

At the application level, security roles are qualified by the display name, if one is specified in the deployment descriptor. At the system level, security roles are optionally qualified with a prefix that is specified in the EJBROLEPRFX system initialization parameter.

A security role with its qualifiers is known as a deployed security role. The mapping of individual users or groups to deployed security roles is carried out in the RACF entity class EJBROLE or the RACF grouping class GEJBROLE.

8.4.1 Using the RACF EJBROLE generator utility

The RACF EJBROLE generator utility (dfhreg) is a Java application program that extracts security role information from a deployment descriptor, and generates an output file containing the RACF commands that define security roles as members of a profile in the GEJBROLE class.

Example 8-1 shows a UNIX System Services (USS) command that invokes the RACF EJBROLE generator utility. In this example,:

- -secprfx PRODCICS specifies the name used to qualify the security role at the system level. The value you specify must match the value of the EJBROLEPRFX system initialization parameter for the CICS system where the security roles will be used.
- PayrollSessionBeanEJB.roles specifies the name of the file to which dfhreg will write its output.
- PayrollSessionBeanEJB.jar specifies the name of the input file which contains the deployment descriptor.

Example 8-1 Executing dfhreg from a USS command line

```
dfhreg -secprfx PRODCICS -out PayrollSessionBeanEJB.roles
PayrollSessionBeanEJB.jar
```

Example 8-2 shows the output file generated by dfhreg.

Example 8-2 PayrollSessionBeanEJB.roles file created by dfhreg

```
/*******************************************************************/
/*                    RACF EJBROLE GENERATOR TOOL                  */
/*******************************************************************/
/* Default enterprise role used to group EJBROLE definitions. */

RDEFINE GEJBROLE DFLTROLE

/* Manager role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.manager)

 /* Team Leader role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.team_leader)

/* Data Entry role */
RALTER GEJBROLE DFLTROLE ADDMEM(PRODCICS.Payroll.data_entry)
```

These RACF commands define the three deployed security roles (PRODCICS.Payroll.manager, PRODCICS.Payroll.team_leader, and PRODCICS.Payroll.data_entry) as members of the profile DFLTROLE in the RACF grouping class GEJBROLE. You can then use the RACF PERMIT command to provide the appropriate users with READ access to the DFLTROLE profile:

```
PERMIT DFLTROLE CLASS(GEJBROLE) ID(user1,user2) ACCESS(READ)
```

Alternatively, you can use the following commands to define the deployed security roles in the EJBROLE entity class:

```
RDEFINE EJBROLE (PRODCICS.Payroll.manager) UACC(NONE)
RDEFINE EJBROLE (PRODCICS.Payroll.team_leader) UACC(NONE)
RDEFINE EJBROLE (PRODCICS.Payroll.data_entry) UACC(NONE)
```

You can then use the RACF PERMIT command to provide a user with access to one security role without giving the user access to the other security roles:

```
PERMIT PRODCICS.Payroll.manager) CLASS(EJBROLE) ID(user1) ACCESS(READ)
PERMIT PRODCICS.Payroll.team_leader) CLASS(EJBROLE) ID(user2)
ACCESS(READ)
PERMIT PRODCICS.Payroll.data_entry) CLASS(EJBROLE) ID(user3)
ACCESS(READ)
```

8.5 Application-managed security

Applications can query the session context object to retrieve security information. Usually, this must be avoided because the objective of EJB is to let the application programmer concentrate on the business logic. However, sometimes it may be necessary.

The EJB 1.1 specification provides the following two methods:

- isCallerInRole(String roleName)

  Returns true if the caller is in the security role specified as the single string argument. Otherwise, false is returned.

- getCallerPrincipal()

  Returns an object of the class java.security.Principal, which may then be used to extract more information such as the distinguished name (DN) that is associated with this session. Figure 8-6 shows a Java code sample for extracting the DN.

```
// obtain the caller principal
   callerPrincipal = ejbContext.getCallerPrincipal();
// obtain the caller principal's name
   callerName = callerPrincipal.getName();
```

Figure 8-6 Extracting distinguished name

When a Java program calls the getName() method, the DN is extracted from the X.509 certificate provided by the client. The DN includes user information such as Common Name, Title, e-mail address, Organization Unit, Organization, Location, State, and Country.

If a certificate is not available, the DFHEJDNX user-replaceable module is invoked to generate the DN, given the following inputs:

- The Common Name, that is, the *user name* associated with the caller's user ID. The user name is a 20-character name of the user obtained from the external security manager (ESM).
- Certain fields from the CICS *client* certificate that is used by outbound SSL and specified in the CERTIFICATE parameter of the CORBASERVER definition. If a CORBAServer certificate is not available, the default certificate provided in the key ring is used instead.

8.6 Design issues

The solution design must consider the security requirements, the performance issues, and the possible implications of flowing IIOP requests through firewalls that may not provide an IIOP capable proxy server.

For a distributed program link (DPL) issued from within CICS, the standard multiregion operation (MRO) security rules apply, including the considerations for link security and equivalent systems.

Related publications

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

IBM Redbooks

For information about ordering these publications, see “How to get IBM Redbooks” on page 252. Note that some of the documents referenced here may be available in softcopy only.

- *CICS Transaction Gateway for z/OS V6.1,* SG24-7161
- *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2*, SG24-6284
- *Implementing CICS Web Services*, SG24-7206
- *z/OS WebSphere Application Server V5 and J2EE 1.3 Security Handbook,* SG24-6086

Other publications

These publications are also relevant as further information sources:

- *CICS Transaction Server for z/OS V3.1 CICS Application Programming Guide*, SC34-6433
- *CICS Transaction Server for z/OS V3.1 CICS Application Programming Reference*, SC34-6434
- *CICS Transaction Server for z/OS V3.1 CICS Resource Definition Guide*, SC34-6430
- *CICS Transaction Server for z/OS V3.1 CICS System Definition Guide*, SC34-6428
- *CICS Transaction Server for z/OS V3.1 Internet Guide*, SC34-6450
- *CICS Transaction Server for z/OS V3.1 RACF Security Guide*, SC34-6454

- *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SC34-6458.
- *WebSphere MQ Security*, SC34-6588
- *WebSphere MQ Using Java,* SC34-6591
- *z/OS V1R7.0 Communications Server: IP CICS Sockets Guide*, SC31-8807

Online resources

The following Web sites are also relevant as further sources of information:

- CICS SupportPacs

  `http://www.ibm.com/software/ts/cics/txppacs`
- Extensible Markup Language (XML) V1.0

  `http://www.w3.org/TR/REC-xml`
- Security on System z

  `http://www.ibm.com/servers/eserver/zseries/security/features.html`

How to get IBM Redbooks

You can search for, view, or download IBM Redbooks, IBM Redpapers, Hints and Tips, Draft Publications, and Additional Materials, and order a hardcopy of IBM Redbooks or CD-ROMs from the following Web site:

ibm.com/redbooks

Help from IBM

IBM Support and Downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Securing Access to CICS Within an SOA

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Securing Access to CICS Within an SOA

Provides information about transforming CICS assets into SOA solutions

Furnishes updates about CICS TS V3.1 and CICS TG V6

Covers CICS Web services and CICS Web support

With the emergence of service-oriented architecture (SOA), the options for accessing the existing IBM Customer Information Control System (CICS) assets have become more varied than ever. With this variety comes the complexity of securing these assets. This IBM Redbook is intended for IT architects who are involved in the process of selecting, planning, and designing a secure SOA solution that makes use of CICS assets.

This book introduces SOA and the options available for transforming CICS assets into SOA solutions. It then discusses the principles of security, followed by the different security technologies.

The book then reviews each technology individually, discussing the security options that are available, the security architectures such as basic authentication, firewalls, and the use of Secure Sockets Layer (SSL) and public key infrastructure (PKI).

IBM

Redbooks

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

SG24-5756-01 ISBN 0738496715