
Janus Network Security Reference Manual



Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, MA 02139

Telephone: (617) 876-6677
FAX: (617) 234-1200
E-mail: support@sirius-software.com
World Wide Web: <http://sirius-software.com>

January 25, 2010

© 2010 Sirius Software, Inc.

Proprietary Notices

The following products:

- *Janus Network Security*
- *Janus TCP/IP Base*
- *Janus Web Server*
- *Sirius Mods*
- *UL/SPF*

are proprietary products of Sirius Software, Inc.:

Sirius Software, Inc.
875 Massachusetts Avenue, Suite 21
Cambridge, Massachusetts 02139
USA

Model 204™ is a proprietary product of Computer Corporation of America:

Computer Corporation of America
200 West St.
3rd Floor West
Waltham, MA 02451
USA

Model 204™ is a registered trademark of Computer Corporation of America.

VeriSign is the trade name and also a registered trademark of VeriSign, Inc.

Contents

Proprietary Notices	ii
Contents	iii
Summary of Changes	v
Sirius Mods Version 7.7	v
Sirius Mods Version 7.3	v
Sirius Mods Version 7.2	v
Sirius Mods Version 6.4	vi
Chapter 1: Introduction to Janus Network Security	1
About This Manual	1
About SSL	2
Encryption	2
Authentication	3
Authentication of the server	3
Authentication of the client	4
When should SSL be used?	4
Threats from company insiders	4
Protecting transmission of passwords	5
Protecting weak links	5
About Your SSL Port	5
Security alternatives	6
Secured and unsecured ports allowed on same server	6
Many alternatives for restricting or permitting access	6
Configuring the server port	7
Defining private key decryption subtasks	7
How a browser should connect	8
Specifying a particular port	8
About Keys and Certificates	8
Public key/private key cryptography	9
Certificates	9
Digital signatures	10
Certifying authorities	10
Self-signed certificates	11
Certificate acquisition	11
Chapter 2: Installing and Configuring Janus Network Security Support for SSL	13
Install Janus TCP/IP Base	13

Install the Sirius Mods	14
Install or update UL/SPF	14
Make an Initial SSL Connection	14
Define and start an SSL port	15
Add appropriate users to the JANSSL subsystem	15
Add web rule to authorize user access to the port	16
Allowing any valid logged-in user	16
Accessing JANSSL through non-SSL ports	16
Making WEBUSER a JANSSL subsystem user	17
Connect to the SSL port from a browser	17
Chapter 3: Implementing a Server Certificate	19
Using the Certificate Management Application	20
Generate a certificate request	21
Storing the private key	21
Encrypting the private key	21
Send the certificate request to a CA	22
Receive the signed certificate	22
Review your certificate	22
Add an intermediate certificate	22
Preparing the SSL Port	23
Defining and starting a port	23
Placing start-up commands in User 0	24
Connecting to the Port	24
Acting as an Organizational Certifying Authority	25
Chapter 4: Verifying a Client Certificate	27
Obtain and Install the Client Certificate	28
Obtain and Load Root Certificates for the Server	28
Configure the Server to Request a Certificate from the Client	29
Example: SSL Port Requires Client Certificate	30
Appendix A: Cryptographic Methods	31
MD5digest function	31
RC4encrypt function	32
SHAdigest function	34
Index	35

Summary of Changes

This section describes significant changes to the documentation. Usually, these changes correspond to enhancements made to the underlying product, although they might be simple documentation improvements.

Sirius Mods Version 7.7

The following changes correspond to changes in *Janus Network Security* since version 7.3:

XML API

- A server's method or \$function call requesting client certificate information may cause an SSL renegotiation in order to request a digital certificate from the client. See [“Configure the Server to Request a Certificate from the Client” on page 29](#).
- The ADDCA utility lets you load Sirius-provided SSL CA certificates (see the note in [“Connecting to the Port” on page 24](#)).

Sirius Mods Version 7.3

The following changes correspond to changes in *Janus Network Security* since version 7.2:

- Documentation added for intermediate certificate support ([“Add an intermediate certificate” on page 22](#)). Support for the feature was pre-existing.
- Individual cryptographic functions made available (see [“Cryptographic Methods” on page 31](#)).

Sirius Mods Version 7.2

The following changes correspond to changes in *Janus Network Security* since version 6.4:

- Multiple additions and revisions to the documentation for Chapter 6, [“Implementing a Server Certificate” on page 19](#).

- Documentation added for client certificate support (“[Verifying a Client Certificate](#)” on [page 27](#)). Support for the feature was pre-existing.

Sirius Mods Version 6.4

The following changes correspond to changes in *Janus Network Security* since version 6.3:

- New security protocol option: TLS (Transport Layer Security)

Developed by the IETF Internet standards group, TLS is a more secure extension to SSL V3, and it is supported for both client and server Janus ports.

General references to SSL in the *Janus Network Security Reference Manual* now imply TLS as well, or they are stated as "SSL/TLS" or as "SSL-like." Explicit references to TLS are included where appropriate.

TLS support introduces only the following visible changes to Janus security implementation:

- The SSLPROT parameter of the JANUS DEFINE command has two new bit settings:
 - ♦ X'04', if a port will support only TLS.
 - ♦ X'07', if a port will support TLS, SSL v3, and SSL v2, and offers them in that order of preference. This is the new SSLPROT default.
- The \$WEB_PROTOCOL web server function may now return "4" (which corresponds to the new SSLPROT value), if TLS is being used for the connection.

Janus Network Security consists primarily of support for the SSL (Secure Sockets Layer) and the TLS (Transport Layer Security) protocols, which provide secure communications for users of Janus products. The TLS protocol, developed by the IETF Internet standards group, is a more secure extension to SSL. This manual will sometimes use "SSL" loosely to refer to SSL (V2 and V3) *and* TLS, and it will also use the terms "SSL-like" and "SSL/TLS."

These protocols are most typically used to encrypt and authenticate the communications of *Janus Web Server* applications. Used by web servers throughout the Internet, SSL and TLS are industry standard approaches to secured communications that enable users to transmit private information, such as credit cards, with the confidence that the encrypted data cannot be intercepted and that the authenticated recipient of the data is the intended one.

Secured communications can be valuable within an organizational intranet as well, protecting the integrity of information transmitted within the network. Just because an intranet protects an organization from attacks by outsiders, it is imprudent to ignore potential attacks by insiders — who may find such information even more valuable than outsiders.

1.1 About This Manual

This manual introduces the key SSL concepts, and it addresses the use of SSL/TLS to provide secure communications for connections to your Janus Server sites via the Internet or via an organizational intranet, as well as for connections to SSL servers from your *Janus Sockets* clients. Once *Janus Network Security* is set up and tested, it operates quietly in the background, and little further effort is required to have fully secured communications.

Generally, *Janus Network Security* is installed as an addition to *Janus Web Server*. It is also possible to install it without *Janus Web Server* (for use with other non-web Janus products). For the sake of simplicity, this manual largely assumes that you will be using *Janus Network Security* with *Janus Web Server*.

This introductory chapter provides a technical background for the rest of the manual, whose chapters guide you through the implementation of *Janus Network Security* at your site:

- [“Installing and Configuring Janus Network Security Support for SSL”](#)
- [“Implementing a Server Certificate”](#)

- “Verifying a Client Certificate”

Notes:

- The appendix, “Cryptographic Methods” on page 31, describes the *Janus SOAP* methods that implement the basic *Janus Network Security* hashing and encryption algorithms.
- For messages related to *Janus Network Security*, please refer to the ***Sirius Messages Manual***.
- Much of the content of this ***Janus Network Security Reference Manual*** is reproduced in HTML pages: The first time you connect to an Online via *Janus Network Security*, a default home page set up by Sirius is displayed. In the introduction on that page, a link (“configure and manage Janus SSL support”) leads to pages that restate much of this document.

1.2 About SSL

SSL, the Secure Socket Layer, is a communication protocol that makes it possible to safely transmit secure information over a non-secure network. SSL accomplishes this primarily by encrypting communications. It can also enable authentication of the server and/or the client to ensure that each party is who they say they are. These three components are described below.

1.2.1 Encryption

The primary means by which SSL accomplishes safe communications is through encryption of the data that is sent between a client or browser and a server. This encryption makes it practically impossible (or at least extremely difficult) for anyone other than the intended recipient of the data to decrypt it, even if a third party has access to the complete contents of the communication. This is a way of defeating “snooping”, the capturing and retrieval of passwords and other sensitive information from data packets transmitted over a network and intended for another machine.

The SSL protocol is typically layered over the TCP protocol and under the HTTP (web) protocol. Information that would normally pass between a web server and browser is encrypted by SSL before being sent over the network via TCP/IP. This means that communications over a public TCP/IP network can be more secure than that over a private VTAM network, because VTAM security depends on the security of the physical network while SSL security is based on mathematics. The cost of this extra security is the CPU processing that SSL requires, especially on the server side.

1.2.2 Authentication

In addition to encryption, SSL also provides “authentication.” Authentication means either or both parties (client and server) can be confident that the other is indeed who they say they are.

1.2.2.1 Authentication of the server

SSL server authentication provides information to the client that can be examined by the end user to verify some basic information about the server.

SSL typically provides the country, state or province, city or town, organization name (Mega Industries, Hard Knocks University, Grand Duchy of Liechtenstein, etc.), organizational unit (Department of Obfuscation, Engineering, MIS, etc.) and a server name (Student Web Server, Parts and Supplies Ordering Server, Sensitive Information Server, etc.).

This authentication defeats “spoofing,” the practice of

1. Setting up a server to intercept messages intended for another server, responding to them in such a way that the end user is convinced he or she is communicating with the correct server
2. Capturing sensitive information such as passwords from the end user.

Spoofers sometimes simply pass information through to the correct server and back from the server to the client, recording sensitive information as it passes through. This technique is sometimes known as a “man-in-the-middle attack.”

SSL authentication defeats all forms of spoofing, because it is practically impossible (or extremely difficult) for a spoofing server to send the correct SSL authentication information to the client.

Server authentication is enabled by way of certificates. Certificates are digital IDs that verify that a server has been certified by a publicly recognized authority as being who they say they are. A certificate is installed in the server and then recognized by client browsers.

Janus Network Security comes with an initial certificate, provided by Sirius specifically for use by your organization (Sirius has certified the organization's identity). You can also arrange to purchase a certificate through widely recognized companies, called Certificate Authorities, if it is important for your users to know that your server's identity has been certified by such an organization.

1.2.2.2 Authentication of the client

SSL can also provide client authentication. Using techniques like those used for server authentication, it is possible for a server to verify the identity (more or less) of the client with which it is communicating, without the use of a password database on the server.

Janus Network Security supports client certificates, which establish the identity of the client. These are acquired like server certificates and installed on client machines — typically browsers but possibly a Janus SSL CLSOCK machine, for example. Not required by most web servers, these certificates are requested and sent during the initial connection negotiation between client and server, or during a subsequent connection renegotiation.

For more information about Janus support for client certificates, see [“Verifying a Client Certificate” on page 27](#).

In addition, client authentication of a sort can be achieved by requiring a user to enter a valid userid and password (as defined in the online in which *Janus Web Server* is running) when connecting to a protected resource.

This optional feature, which can be set for all or for specific applications on a server, is described in the next chapter. Because of SSL/TLS encryption of communications, such passing of userids and passwords over the network is secure.

1.2.3 When should SSL be used?

The obvious answer is whenever secure data is to be transmitted. Unfortunately, a myth has developed around SSL and security in general, that one need only be concerned about it when communicating over the Internet, home of socio-pathic hackers and criminal geniuses whose days are spent trying to wreak havoc and steal information from your system.

1.2.3.1 Threats from company insiders

The reality is that there is generally more opportunity and motivation to steal information inside a company than outside. Who, after all, is more interested in personnel records than corporate insiders? Who is more interested in the personal e-mail of a supervisor than the supervisor's subordinates?

Furthermore, intranets tend to be made up of one or more LAN segments where packets are transmitted freely for all to see. This is a much easier environment in which to steal data by capturing packets than the Internet, where it is now fairly difficult to capture packets not intended for your local network.

Finally, it doesn't take a criminal genius or a hacker to steal data sent over a local network. Any moderately skilled individual (someone who can use a spreadsheet)

armed with a network diagnostic tool (which incidentally has a legitimate purpose) can capture any and all information sent over a local network.

1.2.3.2 Protecting transmission of passwords

The bottom line is that SSL should be used on **any** connection over which sensitive data is to be transferred. Perhaps the most important sensitive data in any *Model 204* system is passwords. Since passwords are used to protect other resources, access to a password will compromise the security of all resources protected by that password. Even worse, certain passwords such as system manager passwords can be used to gain access to any resource in a *Model 204* region.

A good rule of thumb, therefore, is that any connection over which a password will pass should use SSL. *Janus Web Server* actually issues a warning if you try to define a JANUS WEB rule that would result in a password being transmitted over a non-SSL connection. This would happen with any JANUS WEB ALLOW rule that has a USER clause in it, since the USER clause indicates a user login, hence userid/password are required for access to the URL specified by the JANUS WEB rule.

1.2.3.3 Protecting weak links

A dangerous fallacy is that if a resource is only slightly private — that is, while you want to limit access to the resource, it's not that big a problem if someone gets access to it — you don't need to use SSL. Remember, even if the resource you are protecting is not that important, the password of the userid accessing that resource is likely to be important. Even worse, once a userid/password is used to access any URL on a non-SSL *Janus Web Server* port, that userid/password combination will be sent by the browser (unencrypted) for all other URLs on that port (just in case these require a login). Thus by allowing a password to be transmitted unencrypted for a single URL on a non-SSL port, that password might be sent, unencrypted, hundreds of times over the network.

Once again, a good basic rule of thumb is that if a login is required to access a resource or URL, **always** force access to that resource or URL to go through SSL.

1.3 About Your SSL Port

This section introduces some of the ways that *Janus Network Security* can be configured to provide different access controls for your users and applications. It briefly describes how to make such configurations, as well as how to make a connection from a web browser.

1.3.1 Security alternatives

Janus Network Security is flexible and able to provide a variety of levels of security to suit your needs. This section describes the different approaches available to you for securing (or not securing) ports for your *Janus Web Server* applications.

Before proceeding, it may be useful to clarify the term “port.” A **port** is a defined entrance into your server, which can permit entry by one or more users. You can have more than one port on a server.

1.3.1.1 Secured and unsecured ports allowed on same server

You can choose to force SSL-protected communications on all ports on your server, or you could enable it only on specific ones. That means you could have a web site which provides some documents that are served only on secured ports while allowing other documents to be served on unsecured ports.

1.3.1.2 Many alternatives for restricting or permitting access

Several choices are available to you in restricting or permitting access to your secured ports, both in terms of applications and users. These choices are set by the configuration of the server port, described later in this document but outlined briefly below.

Controlling access by application on a secured port:

On a given secured port you can decide whether to require SSL-enabled communications for any or all applications served on that port. That means you can implement tighter security for one application that you may for another.

Controlling access by user validation:

For any or all applications, you can limit access by requiring that users log in with a valid userid and password as defined in the *Model 204* online in which *Janus Web Server* is running. (Note that you can instead permit access by users without userids, as described below.)

By configuring the port to require client validation, the user would be prompted when connecting to a web page or application defined for such access control. This choice of access control would mean that only users who are defined in the online and who log in with a valid password would be able to access the application.

Restricting access to one, several, or a group of users:

Having been validated, access can then further be restricted to one user, a list of them, or a defined group of them. The group of users would be defined in the configuration of *Janus Web Server*.

Permitting access to all web users:

Alternatively, you can permit any and all users on the Internet (or intranet) to connect to an application, without requiring any log in at all. This approach is common on most commercial web sites using SSL for secured communications. In such a case, communications is still secured, but you do not expect to know in advance the identity of all users who may be connecting to your server (nor must you create logon IDs and passwords for them in the *Model 204* online.)

1.3.2 Configuring the server port

The use of SSL is a characteristic of a TCP/IP server port. That is, either all communications to a given server port uses SSL, or none of it does. (As described above, you can define multiple ports on a server, so some can be secured while some are not.) The use of SSL on a port is specified on the JANUS DEFINE subcommand.

Janus commands are entered in the *Model 204* Online in which *Janus Web Server* is running, generally by a system manager. They define how a server port is to be configured, and they are described further in the ***Janus Web Server Reference Manual***.

For example, the following command defines a port that is to use SSL on port number 443:

```
JANUS DEFINE WEBSSL 443 WEBSERV 10 SSL JANSSL MYCERT
```

The **SSL** keyword is followed by the name of a file and a procedure in that file that contains a **certificate** and **private key** to be used by SSL for the port being defined.

Further information on port configuration is provided in [“Define and start an SSL port” on page 15](#).

More information about certificates and private keys is available in [“About Keys and Certificates” on page 8](#).

1.3.3 Defining private key decryption subtasks

If you use SSL ports under MVS, you may want to configure one or more private key decryption subtasks. These are real MVS subtasks which are devoted to decrypting SSL private keys, a CPU intensive process. Moving this work to real subtasks increases *Model 204* maintask availability for other work. These subtasks do not depend on the MP/204 feature, and they can be used whether or not you have MP/204.

For a complete description of the decryption subtask parameters MAXRDS and ACTRDS, see the ***Sirius Mods Command and Parameter Reference Manual***.

1.3.4 How a browser should connect

Browsers decide whether to use SSL based on the scheme or service part of the URL to which they are connecting. More specifically, if the service indicates *http* as in

```
http://sirius-software.com/football/scores
```

then SSL will not be used. If the service indicate *https* as in

```
https://sirius-software.com/personnel/salaries
```

then SSL will be used.

This does not mean that a user can gain unsecured access to a secured resource simply by specifying *http* rather than *https*. On the contrary, if the browser and the server are in disagreement over whether SSL is to be used, a communication error will occur.

When a web browser connects to an SSL-secured port, the browser displays an image of a connected key at the bottom of the screen. Depending on the browser and how it has been configured, the user may also receive a message upon transmitting data to the server that indicates that their communication is secured.

1.3.4.1 Specifying a particular port

When a port is defined for either secure or non-secure web server access, a number is associated with it, called the port number. The default port number for non-SSL web (HTTP) service is 80. When attempting to connect to a web server at its default address, you need not specify the number in the URL. To access a non-SSL web server port defined with a number other than 80, you must indicate the number of that port on the URL.

Similarly, the default port for SSL web (HTTPS) service is 443. Just as it is possible to make a connection to a specific port when not using SSL, it is also possible to do so using SSL. For example, the following URL indicates an attempt to open SSL communications on port 1776:

```
https://sirius-software.com:1776
```

1.4 About Keys and Certificates

In order to provide for secure communications and authentication, SSL employs several features, including public and private keys for encryption, and certificates and digital signatures for authentication. Each of these is described in this section, which concludes with a discussion of how to obtain and use certificates.

1.4.1 Public key/private key cryptography

An important component of SSL security is “public key/private key” cryptography. Such cryptography depends on the existence of pairs of large numbers that can be used as inputs to a formula or program for the purposes of encryption and decryption. The actual nature of the formula and numbers is beyond the scope of this document. Furthermore, knowledge of the details of the public key/private key algorithms is unnecessary for use of SSL.

Some important features of public and private keys are useful to keep in mind, however:

- Something encrypted with a public key can only be decrypted with the private key.
- Something encrypted with a private key can only be decrypted with the public key.
- It is (practically) impossible to derive the private key from the public key.
- If something correctly decrypts with the public key, it must have been encrypted with the private key.

Given these four features, it is then possible to assert that the holder of a private key can freely distribute the corresponding public key, and can be sure that only he or she could decrypt anything encrypted with that public key. This, of course, depends on the private key being protected from access by anyone other than its owner.

1.4.2 Certificates

One application of public/private key cryptography is the creation of certificates, which authenticate the authority of a client or server. A certificate is the formatted structure in which a public key is typically distributed. This structure also contains some basic information about the holder of the private key associated with the public key.

Server certificates used by SSL typically contain the country, state or province, locality (city, town, village, etc.), organization, and organizational unit of the holder of the private key. In addition, SSL server certificates contain the TCP/IP host name of the server that holds the private key.

The information in a server certificate can normally be viewed in a client application such as a browser, so an end user can be sure that he or she is communicating with the correct server.

Note: An SSL client will automatically verify that the host name in a server certificate matches the name of the host to which it is connecting.

A server may also request a sender's **client certificate** for authentication. The certificate contains information about the user, in addition to a unique private-public key pair. Client certificates, like server certificates, are obtained from a trusted source, and

they are stored on the sending computer and accessible from some certificate management tool.

1.4.2.1 Digital signatures

One issue that arises is how to be sure that the information in a certificate does, in fact, correctly reflect the holder of the private key. It would seem that anyone could create a private and public key, package the public key in a certificate with phony information, and pass his or herself off as another server owned by another company. This problem is virtually eliminated by the use of **digital signatures**.

A digital signature for a certificate is a hash of the certificate encrypted with a private key. Digital signatures have the following characteristics:

- The hash function must be well-known and not reversible. That is, it should not be possible to start with the hash and work back to a certificate. The well-known hash function usually used in SSL is MD5.
- The public key associated with the private key used to create the digital signature must be known by the receiver of the certificate. If a certificate is to work with many different clients, it is helpful for this public key to be “well known.”
- The private key used to create the digital signature must be well protected, so only the private key's holder can digitally “sign” a certificate.
- The holder of the private key used for signing (sometimes called a **certifying authority**, or CA) makes an effort to verify that the information in the certificate correctly identifies the holder of the associated private key. For example, a CA should verify that the organization name in a certificate actually matches the name of the organization that holds the associated private key.

1.4.2.2 Certifying authorities

There are several certifying authorities whose public keys are shipped with most browsers. Chief amongst these is VeriSign, Inc., at <http://www.verisign.com>. A certificate digitally signed by VeriSign has these benefits:

- The holder of the private key associated with the certificate is almost certainly correctly identified by the certificate.
- Almost every SSL client will be able to verify the signature on the certificate (because the VeriSign public key is so widely known).

Most browsers have a mechanism for adding a new CA to their database of valid CAs. This makes it possible to use certificates signed by non-standard CAs.

1.4.2.3 Self-signed certificates

Some browsers accept a server's "self-signed" certificate. That is, the private key used to sign the certificate is the private key associated with the public key in the certificate.

Self-signed certificates have some problems associated with them:

- Since the certificate is self-signed, there is no way to be certain from just the certificate that the certificate correctly describes the holder of the associated private key. Fortunately, there are often other means to validate this information.
- Some browsers don't accept self-signed certificates, and those that do, often require extra user interaction to use the self-signed certificate (as they should).

Nevertheless, non-standard CAs and self-signed certificates can be useful for testing and for establishing secure connections before a certificate signed by a standard CA has been obtained.

For example, the *Janus Web Server* application for generating a request for a certificate signature (to VeriSign, Inc., or some other standard certifying authority) should be run in secure mode, that is, over an SSL connection. What certificate should be used to secure the connection the first time this application is run? The answer is a self-signed certificate provided by Sirius Software with the *Janus Network Security* support. (This is also referred to elsewhere in this document as the "internal certificate".)

1.4.2.4 Certificate acquisition

Getting a valid certificate and private key typically involves a multi-step process like the following:

- You generate a private key and an associated certificate request.
- You send the certificate request to the certifying authority (CA).
- The CA digitally signs the certificate and sends it back to you.
- You receive the signed certificate, associating the signed certificate with the private key.

With its Certificate Management Application, described in detail in ["Implementing a Server Certificate" on page 19](#), *Janus Network Security* helps you through this process. It provides an application for generating private keys and certificate requests, and an application for receiving signed certificates from a certifying authority. It also provides an application for signing certificate requests.

The last of these applications is useful if a site wants to have a local certifying authority. For more about this, see ["Acting as an Organizational Certifying Authority" on page 25](#).

Installing and Configuring Janus Network Security Support for SSL

Before you can start using secured communications with *Janus Network Security*, preliminary tasks must be completed. Some of these tasks may be performed by different people, but they should occur in the following sequence:

1. Install *Janus TCP/IP Base*
 - a. Install the *Sirius Mods*, which contains *Janus Network Security* support.
 - b. Install or update *UL/SPF*.

Step “1.” would probably be done by the person responsible for installing and supporting *Model 204* at your site. If you are already running *Janus Web Server*, this step involves some minor updates for support of *Janus Network Security*.
2. Make an initial connection to a port that uses the Sirius-provided internal certificate.
 - a. Define and start the SSL port.
 - b. Give appropriate user(s) correct privileges.
 - c. Download internal certificate as CA to browser.

In step “2.” you test the success of your first connection, using the Sirius-provided internal certificate (discussed in “[Self-signed certificates](#)” on page 11).

Each of these steps is further described below.

2.1 Install Janus TCP/IP Base

As mentioned earlier, *Janus Network Security* is generally installed as an addition to *Janus Web Server*. It can also be installed without it, for use by other non-web Janus products. In either case, some of the steps in the following subsections may have already been completed. You should confirm each one before proceeding to the next step.

2.1.1 Install the Sirius Mods

The *Sirius Mods* are a combination of features which provide a base of support for most Sirius products, and they must be installed prior to proceeding.

You can confirm whether the *Sirius Mods* are installed and at the current release by entering the following command at the *Model 204* command line:

```
SIRIUS
```

This will display various information regarding the configuration of Sirius products in this online, including the current release of the *Sirius Mods*.

2.1.2 Install or update UL/SPF

UL/SPF is something of a misnomer: it encompasses not only some applications that are part of the *UL/SPF* suite but also all *Model 204* files that are required by any Sirius Software products. A base portion of *UL/SPF* is required for *Janus Network Security*, so *UL/SPF* must be installed prior to proceeding. More information is available in the ***UL/SPF Installation and Maintenance Guide***.

Note: As part of *UL/SPF* installation you should allocate and initialize a file called JANSSL, which has the correct characteristics for storing an SSL private key. The JANSSL file should be a minimum of 1050 pages; the *UL/SPF* SIRIUS file contains a sample CREATE.JANSSL deck.

UL/SPF installation also creates a subsystem called JANSSL, which provides components for the certificate management application. The SIRIUS file that contains the definition is SUBSYS.DEF.JANSSL. As described later ([“Add appropriate users to the JANSSL subsystem” on page 15](#)), you may need to update the subsystem.

2.2 Make an Initial SSL Connection

In order to implement and test the success of your first SSL connection, you can use the Sirius-provided internal certificate, which was distributed with the *Janus Network Security* tape (and discussed further in [“Self-signed certificates” on page 11](#)). Ultimately, you will probably want to arrange to receive a certificate from a commercial certifying authority, as described later in [“Implementing a Server Certificate” on page 19](#).

The steps in the subsections that follow describe how to set up and connect to a *Janus Web Server* server port. If you are configuring a Janus CLSOCK SSL port, for example, the same sequence of steps apply (except for the last subsection), but the keywords WEB and WEBSERV in the JANUS command examples would be replaced as necessary by CLSOCK.

2.2.1 Define and start an SSL port

Your SSL port definition indicates that connections to the port are to use the Sirius-provided internal certificate. In the server Online, a user with system manager or system administrator privileges must issue a command like:

```
JANUS DEFINE SSLWEB 443 WEBSERV 5 SSL *
```

where:

- `SSLWEB` can be replaced by an arbitrary port name.
- `443` can be replaced by an arbitrary port number (although 443 is the default for HTTPS).
- `5` can be replaced by a different maximum thread value.
- `SSL *` indicates that the internal certificate and private key is to be used. When a certificate from a certifying authority is being used, you replace the `*` with the location and name of the procedure that contains the certificate, as shown in [“Defining and starting a port” on page 23](#).

For more information about the `JANUS DEFINE` command, see the *Janus TCP/IP Base Reference Manual*.

To start the port, enter:

```
JANUS START SSLWEB
```

where `SSLWEB` is the port name specified on the `JANUS DEFINE` command. The port is started and SSL is enabled.

2.2.2 Add appropriate users to the JANSSL subsystem

When the *UL/SPF* installation is done, the user doing the install is given privileges to run JANSSL. If the user that did the *UL/SPF* installation is not the one that will manage certificates and private keys, the latter user must be added as a JANSSL subsystem user (probably using `SUBSYSMGMT`). For example, if MARY installed *UL/SPF* but CURTIS is going to manage certificates, CURTIS must be added as a JANSSL subsystem user.

In addition, while using `SUBSYSMGMT`, make sure that (for *UL/SPF* 6.8 or greater) the JANSSL subsystem communication global is `NEXT`, the exit value is `LOG`, and the procedure file is `SIRIUS`.

2.2.3 Add web rule to authorize user access to the port

When the web port using the internal private key is defined, *Janus Web Server* rules are automatically created so that the defining user can access any resource on the SSL port. If the user that issued the JANUS DEFINE command (in [“Define and start an SSL port” on page 15](#)) is not the one that will manage certificates and private keys, a rule must be added to allow the latter user to access resources on the SSL port.

For example, if MORT issued the JANUS DEFINE command for the SSL port but TAMARA is going to manage certificates, add a rule like the following to the SSL port definition to allow TAMARA to access resources on that port:

```
JANUS WEB SSLWEB ALLOW * USER TAMARA
```

This is the best way to protect the server. The following subsections describe three other ways that you can authorize users to access the SSL port. Though you may be tempted to use them, ***each of the following is discouraged***, as described below. For additional comments about access, see [“Many alternatives for restricting or permitting access” on page 6](#).

2.2.3.1 Allowing any valid logged-in user

The preceding paragraphs described how to add a web server rule to authorize an individual user to access the secured port. An even simpler approach is to modify the rule as follows to allow any valid userid to access data on the SSL port:

```
JANUS WEB SSLWEB ALLOW * USER *
```

Although it is not recommended, note that this rule does *not* expose a security hole: without being a JANSSL subsystem user, there is not much a user can do on the SSL thread as shipped by Sirius. (Remember, JANSSL is the subsystem installed with *Janus Network Security* that provides support for the certificate management application.)

However, this degree of access might become an issue as you add your own applications to the SSL port, at which point it is up to you whether to enforce security through web rules, subsystem access rules, or internal application checks. The appropriate security mechanism for your site in this case is beyond the scope of this document.

2.2.3.2 Accessing JANSSL through non-SSL ports

It is possible to access the JANSSL subsystem through a non-SSL port by specifying a rule such as this:

```
JANUS WEB WEBPORT ALLOW /JANWEB/JANSSL* USER *
```

where `WEBPORT` is *not* your Janus SSL port. While possible, this is **strongly discouraged**, because the above web rule will result in unencrypted *Model 204* passwords being sent over the network as JANSSL verifies access privileges.

The only reason to allow access to JANSSL like this might be that there is local network security that makes you feel confident that this is not a problem, and for some reason, it is impossible to get your browser working with the internal Janus certificate.

2.2.3.3 Making WEBUSER a JANSSL subsystem user

Perhaps the **worst way** to provide access to the certificate management application is one of the following:

- Define as a JANSSL subsystem user the default *Model 204* account ID (normally WEBUSER) that is used for logons of URLs that require no userid.
- Make the JANSSL subsystem public.

These approaches let anyone with a browser generate (and overlay) private keys and certificates in the JANSSL file. Security is completely gone, and you might as well not use SSL. You could perhaps make this somewhat secure by limiting access to certain URLs based on IP address, but this is still considered risky from a security perspective.

2.2.4 Connect to the SSL port from a browser

Having defined and configured an SSL port, you can now attempt a connection. As described in [“How a browser should connect” on page 8](#), the difference between connecting to an SSL-secured port and an unsecured one is in the designation of the service in the URL sent from the browser.

To reach a secured port, you simply specify "https" rather than "http" in the URL that points to your server.

If you chose to define the port with a number other than the default of 443, you must specify that number as well. For example, for a site that is testing the secured port 8443, specify:

```
https://www.yourdomain.com:8443
```

If you have yet to generate an internal certificate, *Janus Network Security* produces by default an out of date Sirius self-signed certificate. In response, your browser will probably initially display one or more notices informing you that the default Sirius certificate has expired, that Sirius is an unknown certifying authority, or that the domain name you specified does not match the name on the certificate. For more information about certificates and certificate authorities, see [“Certificates” on page 9](#) and [“Certifying authorities” on page 10](#).

At the browser, you can view the certificate for supporting evidence that it is indeed from Sirius, then choose to accept it. Next, you are prompted for your *Model 204* userid and password.

Finally, accessing this SSL port for the first time invokes the Default Janus Web Home Page, set up for you by Sirius. You are taken to this page as the result of default JANUS WEB ON rules supplied with the *Janus Web Server* (and viewable via the JANUS DISPLAYWEB command). You can change the default page by defining a JANUS WEB ON rule for your secured port that points to another page (see the ***Janus Web Server Reference Manual*** for more information).

On the default home page, a link in the introduction (“configure and manage Janus SSL support”) points to a page that describes Janus SSL and which restates much of the document you are now reading. This link and those in the Sirius certificate application are also resolved by default JANUS WEB ON rules.

The following chapter, [“Implementing a Server Certificate” on page 19](#), describes how to get an SSL certificate for your site from a certifying authority. If you want to generate an update to the default Sirius internal certificate, that is, generate a self-signed certificate that will bear your own site's current information and date, you also use the request form described in the next chapter.

Implementing a Server Certificate

Ultimately, you will probably want to arrange to receive a certificate from a commercial certifying authority. Certificates and certifying authorities are discussed in some detail in [“About Keys and Certificates” on page 8](#).

The process of getting a valid certificate and private key combination usually involves these steps:

1. Generate a private key and an associated certificate request. The certificate request is basically a certificate without a digital signature. The private key should be stored in a secure location and (preferably) encrypted.
2. Send the certificate request to the certifying authority (CA). This could be done via e-mail, fax, or simply moving files or procedures around on a machine.
3. The certifying authority digitally signs the certificate. Presumably, the certifying authority verifies that the information in the certificate is correct and valid, but this is not strictly necessary.
4. The certifying authority sends the digitally signed certificate back to the holder of the private key.
5. The holder of the private key receives the signed certificate. This usually involves associating the signed certificate with the private key in some way. With *Janus Network Security*, the private key and the signed certificate must be placed into the same procedure.

When generating a certificate request for a CA, you use separate forms from the Janus Certificate Management Application to accomplish steps 1, 2, and 5. When generating a self-signed certificate, all five steps are performed by the same server, so they are combined into a single step from the perspective of the end user.

This chapter discusses how to use the Sirius-provided Certificate Management Application to get either a commercially-signed or a self-signed certificate, and how to set up the web port that uses the certificate at your site.

3.1 Using the Certificate Management Application

This section describes how to use the Janus certificate management application (CMA) to get a signed certificate. The application can generate a certificate that you send to a CA for signing, or it can generate a certificate that it simultaneously signs for you. You may want to use the latter, self-signed, certificate ([“Self-signed certificates” on page 11](#)) for testing purposes.

You access the CMA from your browser with a URL that will cause you to be logged in as a JANSSL subsystem user:

1. Access the Sirius default home page via the SSL port set up to use the Sirius internal certificate (as described in [“Connect to the SSL port from a browser” on page 17](#)):
2. On the Janus default home page, follow the “configure and manage Janus SSL support” link in the introduction.

This brings you to the “SSL Configuration and Maintenance” page, which is a detailed summary of Sirius SSL support (it contains much of the same content as this ***Janus Network Security Reference Manual***). The upper left of this page contains a list of links to the activity pages of the certificate management application:

Create a certificate request

Invokes a form that generates either a certificate request for a CA or a self-signed certificate. If a request for a CA, the output is two procedures: one contains the certificate request, and one contains your private key. If for a self-signed certificate, the output includes the two procedures just mentioned as well as a third procedure which contains the certificate itself.

Receive a signed certificate

Invokes a form that receives a signed certificate when it is sent back to you from a certifying authority(CA). This form is not needed for a self-signed certificate.

Sign a certificate request

Invokes a form that appends *your site's* digital signature to the certificate; it is designed to help you act as an organizational certifying authority.

Note: You do **not** use this form in the exercise in this chapter (port with CA-signed certificate). For more information, see [“Acting as an Organizational Certifying Authority” on page 25](#).

Manage private key and certificate procs

Invokes a form that lets you examine (as well as rename or delete) the certificates and keys you receive and create. This page also lets you append an **intermediate certificate**, should that be necessary (see [“Add an intermediate certificate” on page 22](#)).

Each of the Janus CMA forms contains its own online Help.

To use the CMA to get a CA-signed certificate, you submit the “Create a Certificate Request” and the “Receive a Signed Certificate” forms (and you may use the “Manage Private Key and Certificate Procedures” form for maintenance). For a self-signed certificate, you submit only the “Create a Certificate Request” form (selecting the “Create self-signed certificate” checkbox).

The rest of this section describes the process you follow to get a CA-signed certificate, supplementing the details of form completion found in the online Help.

3.1.1 Generate a certificate request

1. On the top left of the SSL Configuration and Maintenance page, select the “Create a certificate request” link.
2. In the “Help” and “Tips” sections of the “Create a Certificate Request” page, generate the request.

Follow the comments that pertain to generating a certificate request for a CA. In addition, make use of the comments in [“Storing the private key”](#) and [“Encrypting the private key”](#), below.

You are returned a “Generated Certificate Request” page that contains your encryption password and your certificate request. On the mainframe, a .PKEY file (private key) and a .CERT file (certificate request) are added to your JANSSL file.

3.1.1.1 Storing the private key

When you generate the certificate request, a private key will also be generated and stored in JANSSL or some other file you designate. JANSSL is recommended, because it is set up to be impossible for an end user to peruse, and it is set up so that the JANSSL subsystem can place private keys there. It is essential that the private key is stored in a file that is not accessible by an average user. In fact, the certificate management system is set up so that even the user that generated the private key cannot determine the actual value of the private key.

3.1.1.2 Encrypting the private key

Regardless of where the private key is stored, it should also be encrypted. It is recommended that you allow the “Create a Certificate Request” application to automatically generate an encryption password (really a key), since this key will be more difficult to guess than a mnemonic password that you might consider. In any case, write the password down, since without it the private key is useless. It is probably best not to maintain it in a file on any computer, unless you are sure that the file is secure.

3.1.2 Send the certificate request to a CA

Once the certificate request and the corresponding private key are generated, the certificate request must be sent to the appropriate certifying authority, such as VeriSign, Inc. You can e-mail or fax to the CA a copy of the certificate request returned by the request-generating application. It is also possible to make a copy of the certificate request in a *Model 204* procedure and somehow send it to the appropriate CA.

3.1.3 Receive the signed certificate

Once a signed certificate is received from the certifying authority, that certificate needs to be associated with the private key used to generate the certificate request. This association is done with the “Receive a Signed Certificate” form.

The simplest way to move the signed certificate received (hopefully via e-mail) from the CA into the “Receive a Signed Certificate” application is to cut and paste the certificate from e-mail into the application's input form. Depending on the e-mail package you are using and on your browser, the cut-and-pasted certificate might seem to line up oddly in the input text area. This should not be a problem: the form does not have to “look neat” in order for the “Receive a Signed Certificate” application to accept the certificate.

When the certificate is successfully received, you get a confirmation page that contains the “signed certificate information” (from your certificate request) that identifies your server to prospective browser clients.

If your CA provided an intermediate certificate, see [“Add an intermediate certificate”](#), below.

If you generated a self-signed certificate, the receiving step is done automatically for you.

You can use the “Manage Private Key and Certificate Procedures” page of the Janus CMA to check that your certificate is stored in the procedure file you designated.

3.1.4 Review your certificate

After storing your signed certificate, you use the “Manage Private Key and Certificate Procedures” page of the Janus CMA to review the certificate information.

3.1.5 Add an intermediate certificate

As of sometime in 2006, most of the prominent CAs require that servers install one or more intermediate certificates instead of the CAs' well-distributed “root” certificates. For example, if you are getting your server certificate from VeriSign, Inc. and your server software supports chaining, you are required to install an intermediate certificate (as of April, 2006).

A well-known CA like VeriSign, Inc. (whose root certificate is trusted across the internet) typically creates an intermediate CA which is used to sign your server certificate, and in some cases, provides an additional intermediate certificate. This CA “chain” is designed to provide a layer or two of protection for the root certificate.

The Janus certificate management application is equipped to receive intermediate certificates. If your CA returns an intermediate certificate in response to your certificate request, you should actually get two certificates (in the simpler case) back from the CA:

1. The primary SSL certificate for your web server (signed by intermediate CA)
2. A certificate for the intermediate CA (signed by the primary CA)

You use the “Receive a Signed Certificate” page (as described in [“Receive the signed certificate” on page 22](#)) to store the primary SSL certificate (item 1, above). As part of the “receive,” the Janus CMA will search its store of intermediate CA certificates for a match to the signer of your primary certificate. If a match is found, your intermediate certificate (item 2, above) is automatically added at the same time. If no match is found, then you use the “Manage Private Key and Certificate Procedures” page to “manually” add the intermediate certificate to the primary certificate.

To manually append an intermediate certificate:

1. From the “Manage Private Key and Certificate Procedures” page, display the stored certificate by identifying the procedure file and any password, then clicking the `Search file` button.
2. Select the prefix of the .CERT file that contains the certificate.
3. Scroll to the “Add an Intermediate Certificate” form, where you can paste the certificate you get from the CA.

3.2 Preparing the SSL Port

Once you have associated the signed certificate with the private key, you are ready to make available a port that uses this certificate and private key. You will also want to consider the benefit and risk of starting your SSL port via the User 0 stream.

3.2.1 Defining and starting a port

To create a Web Server port that uses the new certificate and private key:

1. Define the port with a command like the following:

```
JANUS DEFINE WEBSSL 443 WEBSERV 10 -  
SSL JANSSL NEWCERT.PKEY
```

where the private key and associated certificate are contained in procedure NEWCERT.PKEY of file JANSSL, and these are the value of the SSL parameter.

2. Start the port by issuing:

```
JANUS START WEBSSL  
mysecretpassword
```

where *mysecretpassword* is the password for the private key discussed earlier in “[Encrypting the private key](#)” on page 21. If you correctly enter the password, the port is started and you may use the signed certificate.

If you enter an incorrect password, you receive the following error message, and the port is not started:

```
MSIR.0378: INVALID RSA PRIVATE KEY
```

3.2.2 Placing start-up commands in User 0

Once you have set up the port definition and start, it will be tempting to place the START command followed by the password into the user 0 CCAIN stream. While there is nothing strictly wrong with this, it does make the user 0 CCAIN stream a key part of your network security, hence this stream should be protected accordingly. If a user could read this CCAIN stream, the user would be halfway to breaking SSL security — they would still have to get access to the private key to be able to decrypt SSL encrypted messages on the network. By being protective of the private key password and the private key itself, you should ensure the efficacy of SSL security.

3.3 Connecting to the Port

To test your SSL connection, connect to the port from your browser as described in “[Connect to the SSL port from a browser](#)” on page 17.

If you are testing with a self-signed certificate, your browser is likely to warn you that the certificate is not from a recognized CA. If this warning becomes bothersome, you can stop it by downloading your certificate to the browser to keep in its database of trusted certificate authorities. Issuing the following browser request sends your certificate to the browser:

```
https://www.yourdomain.com/JANWEB/DOWNCA?JANSSL/mycertprocprefix
```

where *mycertprocprefix* is the prefix of the .CERT file that contains your certificate (and JANSSL is presumably the procedure file). The browser's security interface prompts you for how to handle the incoming certificate.

You may still get an additional warning if *www.yourdomain.com* does not exactly match the host name you specified on the certificate request form.

Note: If you are connecting to the server port from a Janus CLSOCK client, the CLSOCK port does not by default load a database of trusted CA certificates like a browser does. The CLSOCK port must explicitly add a root certificate from the CA that signed the server certificate. The root certificate verifies the server certificate.

One way to get a root certificate for a CLSOCK port is to:

1. Connect to the secure port with a browser.
2. Select the options to examine the server certificate and determine its Issuer.
3. Export the appropriate certificate and upload it to your Online.

After you obtain a CA root certificate, you use a JANUS ADDCA command (described in detail in the *Janus TCP/IP Base Reference Manual*, and shown in an example for authentication of a client “[Example: SSL Port Requires Client Certificate](#)” on page 30).

CLSOCK ports under *Sirius Mods* version 7.7 or higher may also take advantage of the User Language ADDCA utility to add one or more of the standard CA root certificates that Sirius pre-loads to the *SIRIUS* procedure file (as of *UL/SPF* 7.3). For more information about the ADDCA utility, see the JANUS ADDCA command description in the *Janus TCP/IP Base Reference Manual*.

3.4 Acting as an Organizational Certifying Authority

There is no further discussion in this manual of what you do if you want to be your own certifying authority, creating and managing certificates for your organization's subscribers. If you want to explore this possibility or need help with some of the issues discussed in this document, please contact Sirius Software technical support.

Security in general and SSL in particular are complex, and a simple misunderstanding can result in a security hole. It is best to talk things through and be sure you are doing the right thing if there is any doubt whatsoever.

Verifying a Client Certificate

Janus Network Security also supports client certificate verification, which lets you restrict server port access to only those clients that present an appropriate client certificate during an initial or a renegotiated SSL “handshake.” The client's certificate and public identity must be valid and must have been issued by a Certificate Authority (CA) whose root certificate has been explicitly added to the server's list of trusted CAs.

This chapter describes steps that supplement the steps described in the previous chapter for setting up a Janus SSL server.

To add client certificate verification to a Janus SSL server:

1. Require (or arrange for) clients to obtain and install a CA-signed client certificate.
2. Obtain root certificates for the server from CAs that provide the client certificates.
3. Configure the Janus SSL server to request or require the client certificate.
4. Add a CA root certificate to the Janus Server port.
5. Optionally, specify an error handler to launch when a required client certificate is not presented.

This chapter describes these steps in greater detail.

For information about methods and \$functions that retrieve the information contained in a client certificate received by a Janus port, see the following:

- The ClientCertificate method (***Janus SOAP Reference Manual***)
- The \$Web_Cert_Info and \$Web_Cert_Levels functions (***Janus Web Server Reference Manual***)
- The Socket class methods CertInfo and CertLevels, and their \$function counterparts \$Sock_Cert_Info and \$Sock_Cert_Levels (***Janus Sockets Reference Manual***)

4.1 Obtain and Install the Client Certificate

Client browsers come with a set of pre-installed root certificates from multiple certifying authorities which they use to authenticate server certificates. The pre-installed certificates do **not** establish the browser user's identity, however, if a server requests client verification. For client verification purposes, browser clients must apply for, obtain, and install a CA-signed certificate that ties the identity of the browser user (the subject of the certificate) to the particular public key in the certificate.

Client certificates are created much like SSL server certificates. You begin by requesting a certificate from a CA. You can use the CA's web site to generate a certificate request, or you can generate a certificate request using a tool like the Janus SSL CMA, which is described in [“Using the Certificate Management Application” on page 20](#).

The certificate request contains the private key that does both of these:

- Determines the public key embedded in the certificate that the CA ultimately returns
- Participates in the CA's digital signature of the returned certificate

Note: *Janus Network Security* servers that request client verification accept only incoming certificates that are signed by a trusted CA.

For a browser client, how you install the client certificate you get from a CA, and where you store the private key you generated, is browser-specific. This document leaves those details to the user. If the client to be authenticated is a Janus CLSOCK client, the details of certificate acquisition parallel those for a Janus server, as described in the previous two chapters.

4.2 Obtain and Load Root Certificates for the Server

The client to be authenticated must present to the server a certificate that is signed by a certifying authority whose certificate is recognized by the server port. The server recognizes only those CAs whose root certificates it has already obtained, stored locally, and added to the port using the JANUS ADDCA command (described in detail in the *Janus TCP/IP Base Reference Manual*). This command must reference the CA's certificate, which is typically loaded into a *Model 204* procedure inside the region (see the code in [“Example: SSL Port Requires Client Certificate” on page 30](#)).

One way to get a CA root certificate for the server might be to locate the certificate in your browser's store, then export it, and upload it to your Online.

Depending on your application, you may need to add certificates from multiple CAs.

Servers under *Sirius Mods* version 7.7 or higher may also take advantage of the User Language ADDCA utility to add one or more of the standard CA root certificates that Sirius pre-loads to the **SIRIUS** procedure file (as of *UL/SPF* 7.3). For more information about the ADDCA utility, see the JANUS ADDCA command description in the ***Janus TCP/IP Base Reference Manual***.

4.3 Configure the Server to Request a Certificate from the Client

Your application must indicate to the client that it wants to examine the client's certificate. You can do so in either or both of the following ways:

- You can configure the Janus port to request or require the client certificate,
- As of *Sirius Mods* version 7.7, you can request a client certificate (after the SSL connection is established) by calling one of the methods or functions that query such a certificate for information. These methods and functions are listed in the chapter introduction [on page 27](#).

The certificate request occurs during a connection handshake or during a handshake renegotiation, and the request is made only once. If you use both the port definition and the method/function calls, the port definition makes the request.

If you decide to use the port definition to request a certificate, specify either of the following JANUS DEFINE command parameters, which are described further in the ***Janus TCP/IP Base Reference Manual***:

- The SSLCLCERT parameter causes the port to request a certificate from the client, but processing continues if no certificate is presented.
- The SSLCLCERTR parameter requires a client certificate, and processing terminates if no certificate is provided.

Note: If SSLCLCERTR is specified on the port definition, you can also specify an exception handler routine that will perform error processing if no client certificate is presented. You call the handler by creating an SSLNOCERTERR rule on either a JANUS ON, TYPE, or REDIRECT command (see [“Example: SSL Port Requires Client Certificate” on page 30](#)).

4.4 Example: SSL Port Requires Client Certificate

The following code defines an SSL server port that requires its clients to present a certificate. An exception handler is called if no client certificate is presented.

The SSLCLCERTR parameter of JANUS DEFINE causes the port to require the client certificate. The exception handler is invoked via the ON rule with the SSLNOCERTERR.

This port definition also requires the following:

- At least one certificate from a certifying authority must be in place on the server. This is accomplished by the ADDCA rules.
- The START command must be followed by the password for the server's private key (discussed earlier in [“Encrypting the private key”](#) on page 21).

```
JANUS DRAIN CLCERTWEB
JANUS DELETE CLCERTWEB
```

```
* Port using SSL security with the session cookie (web port)
```

```
JANUS DEFINE CLCERTWEB 9733 WEBSERV 10 TRACE 1 UPCASE      -
      OBSIZE 10240 RBSIZE 10240 IBSIZE 4096 HTTPVERSION 1.1 -
      SSL JANSSL TM2008.PKEY SSLCACHE 100 SSLBSIZE 32767   -
      OPEN FILE MYPROC MAXTEMP 9000 TIMEOUT 180          -
      NOAUDTERM COMPRESS 0 KEEPALIVE 60 SSLCLCERTR
```

```
* Certifying Authority certificates to authenticate clients
```

```
JANUS ADDCA CLCERTWEB MYPROC SECURESE.CERT
JANUS ADDCA CLCERTWEB MYPROC THAWTE.CERT
JANUS ADDCA CLCERTWEB MYPROC VERIJUNK.CERT
```

```
JANUS WEB CLCERTWEB  DISALLOW *
JANUS WEB CLCERTWEB  ALLOW * USER *
```

```
JANUS WEB CLCERTWEB ON SSLNOCERTERR OPEN FILE MYPROC      -
      CMD 'INCLUDE MISSING_CERTIFICATE_ERROR'
```

```
JANUS START CLCERTWEB
sslpwdrequiredbyport
```

APPENDIX A *Cryptographic Methods*

The object-oriented methods described in this appendix share the code of the underlying functions employed by the *Janus Network Security* Certificate Management Application. These methods are members of the *Janus SOAP* String intrinsic system class. As such, they are available for all *Sirius Mods* customers, in addition to customers of *Janus Network Security*.

For more information about using *Janus SOAP* class methods, see the ***Janus SOAP Reference Manual***.

A.1 MD5digest function

This function returns the 16-byte (always) binary string that is the MD5 digest (hash) of the input string.

```
%outStr = string:MD5digest
```

MD5digest syntax

Syntax Terms

%outStr A string variable to receive the MD5 digest of the input string.

string The string to which the method is applied.

Usage Notes

- MD5 (Message-Digest algorithm 5) is a well-known cryptographic hashing function which is used in the *Janus Network Security* product for digital signatures. A complete explanation of MD5 hashing can easily be found on the internet.
- SHAdigest (“[SHAdigest function](#)” on page 34) returns the SHA digest of the input string, and RC4encrypt (“[RC4encrypt function](#)” on page 32) returns the RC4 encryption of the input string.

Examples

The 16-byte MD5 hash of a string is typically expressed as a 32-digit hex value. In the following example, the output string from the MD5digest method is converted to hex using the Intrinsic method StringToHex (described in the *Janus SOAP Reference Manual*).

```
begin
%string is Longstring
%string = 'this is a test'
Print 'this is a test = ' %string:md5digest:stringtohex
end
```

The result is:

```
this is a test = 9034E10B7993EB846B6D127070E71E25
```

A.2 RC4encrypt function

This function returns a binary string that is the input string encrypted or decrypted with the specified RC4 encryption key. The length of the returned string is the same as that of the input string.

RC4decrypt is a synonym for RC4encrypt.

RC4 is a two-way, or symmetric, cipher (algorithm), so encrypting a string with a key and then encrypting the result of that encryption with the same key produces the original string. That is, if `%result = %string:rc4encrypt(%key)`, then `%result:rc4decrypt(%key)` should always return the initial `%string` value.

```
%outStr = string:RC4encrypt(%key)
```

RC4encrypt syntax

Syntax Terms

%outStr A string variable to receive the encrypted or decrypted input string. Its length is the same as `%key`.

string The string to which the method is applied.

%key A string variable whose value is used to encrypt or decrypt the input string. The key is transformed and then combined with the input string.

This key value must not be null nor longer than 255 bytes. (RC4 keys are rarely longer than 64 bytes).

Usage Notes

- You are **not** prevented from creating confusion by encrypting with RC4decrypt and decrypting with RC4encrypt.
- You can use the RC4decrypt synonym to “document” that you are decrypting rather than encrypting in a call.
- The `%key` value you provide is (internally) concatenated with itself until it reaches 256 bytes and becomes the “true” key that participates in the encryption algorithm. A consequence of this is that a key that consists of text that repeats will produce the same encryption result as the text alone (without its repetitions).

For example, the eight-byte key `AAAAAAAA` is no stronger than the one-byte key `A`, so it is not very secure. Similarly, the ten-byte key `ababababab` is no stronger than the two-byte key `ab`.

- RC4 is the default stream cipher used in *Janus Network Security*. A complete explanation of RC4 encryption can easily be found on the internet.
- MD5digest (“[MD5digest function](#)” on page 31) and SHAdigest (“[SHAdigest function](#)” on page 34) are cryptographic hash functions that operate on the input string.

Examples

In the following example, the output string from the RC4encrypt method is assigned to a variable, converted to hex using the StringToHex Intrinsic method (described in the *Janus SOAP Reference Manual*) to reveal its non-displayable characters, then decrypted to return the original input string:

```
begin
%result is Longstring
%result = 'this is a test':rc4encrypt('key')
Print 'Result is: ' %result:stringTohex
Print 'Initial input is: ' %result:rc4decrypt('key')
end
```

The result is:

```
Result is: E15655DAC416D10ACB3730FA22D2
Initial input is: this is a test
```

A.3 SHAdigest function

This function returns the 20-byte (always) binary string that is the SHA digest of the input string.

```
%outStr = string:SHAdigest
```

SHAdigest syntax

Syntax Terms

%outStr A string variable to receive the SHA digest of the input string.

string The string to which the method is applied.

Usage Notes

- SHA (Secure Hash Algorithm) is a set of cryptographic hashing functions; SHAdigest provides SHA-1, the most commonly used. A complete explanation of SHA hashing can easily be found on the internet.
- MD5digest (“[MD5digest function](#)” on page 31) returns the MD5 digest of the input string, and RC4encrypt (“[RC4encrypt function](#)” on page 32) returns the RC4 encryption of the input string.

Examples

The 20-byte SHA hash of a string is typically expressed as a 40-digit hex value. In the following example, the output string from the SHAdigest method is converted to hex using the StringToHex Intrinsic method (described in the *Janus SOAP Reference Manual*).

```
begin
%string is string len 24
%string = 'such a simple test'
print 'such a simple test = ' %string:shadigest:stringtohex
end
```

The result (displayed on two lines) is:

```
such a simple test =      -
BC38AA2D6769639946806616C14AF0C69477AABE
```

Index

A

ADDCA utility, User Language ... 25, 28
 Authentication, client ... 4, 27
 Authentication, server ... 3, 25

C

Certificate
 client ... see “Client certificate”
 intermediate ... see “Intermediate certificate”
 internal ... see “Internal certificate”
 pre-loaded ... 25, 28
 self-signed ... see “Internal certificate”
 server ... see “Server certificate”
 Certificate management application ... 11, 20
 Certifying authority (CA) ... 10
 Certifying authority, local ... 11, 20, 25
 Chained certificate ... 23
 Client authentication ... 4, 27
 Client certificate ... 4, 9, 27
 information, methods for retrieving ... 27

D

Digital signature ... 10

H

Home page, Janus Web default ... 18

I

Intermediate certificate ... 20, 22-23
 Internal certificate ... 11, 13-14, 20, 24

J

JANSSL file ... 14
 JANSSL subsystem ... 14, 16
 JANUS ADDCA command ... 25, 28
 Janus Web home page, default ... 18

K

Keys, encryption ... 8
 See also Private Key; Public Key

M

MD5digest function, String Intrinsic class ... 31
 Methods, cryptographic Intrinsic ... 31

P

Password, private key encryption ... 21, 24, 30
 Port, SSL ... 6
 connection to ... 17, 24
 Janus CLSOCK ... 14, 25, 28
 Janus definition ... 15, 23, 30
 Private key ... 9
 Public key ... 9

R

RC4decrypt function, String Intrinsic class ... 32
 RC4encrypt function, String Intrinsic class ... 32
 Root certificate ... 23, 28

S

Self-signed certificate ... see “Internal certificate”
 Server authentication ... 3, 25
 Server certificate ... 3, 9, 19
 SHAdigest function, String Intrinsic class ... 34
 Spoofing ... 3
 SSL (Secure Socket Layer)
 concepts ... 2
 port ... see “Port, SSL”
 SSL parameter, JANUS DEFINE ... 24

T

TLS (Transport Layer Security) protocol ... 1

W

WEBUSER userid ... 17