HP Secure Web Server

_____Based on Apache _____

SSL User Guide

© 2005 Hewlett-Packard Development Company, L.P.

HP Secure Web Server

_____*Based on Apache*_____

SSL User Guide

November 2005

HP Secure Web Server for OpenVMS

Distribution restrictions

Customer agrees that he/she is not prohibited by the U.S. or other government export control regulations from receiving this software or technical data.

Copyright and trademark information

© 2005 Hewlett-Packard Development Company, L.P.

Apache is a trademark of the Apache Software Foundation.

Netscape Navigator and Netscape Communicator are trademarks of Netscape Communications Corporation.

Internet Explorer is a trademark of Microsoft Corporation.

All other product names mentioned herein may be trademarks or registered trademarks of their respective companies.

Table of Contents

SSL Setup Information

Documentation

Comprehensive usage information for working with SSL is available in the *HP Secure Web Server* SSL User Guide.

This SSL Setup Information is intended to supplement the general Release Notes and the Installation and Configuration Guide for *SWS*.

SSL files

HP Secure Web Server includes two modules for its SSL functionality. These are **OpenSSL** and **mod_ssl**.

Mod_ssl integrates OpenSSL with a set of source patches for Apache called the Extended API (EAPI). These components are included and automatically installed in *SWS*: the OpenVMS implementation of Apache with mod_ssl.

After installing

After installing *HP Secure Web Server*, additional steps are performed automatically for you by running the configuration utility.

```
$ @SYS$MANAGER:APACHE$CONFIG.COM
```

This includes creating a self-signed server certificate and installing it. SWS will not run without a server certificate that is valid for your system. You may want to view the contents of this file using the OpenSSL Certificate Tool before starting the server.

Configuration Options

During the configuration procedure, you have the option to enable or disable SSL (see Disabling SSL *below*) and to add optional command-line arguments to the server.

To enable SSL, choose the default response of "Yes":

```
Do you want to enable the security features provided by MOD_SSL?
If so, the server will support the HTTPS (HTTP over the Secure
Socket Layer) protocol.

Enable MOD_SSL? [YES]
```

The optional command-line arguments enable you to make settings in the main configuration file (`HTTPD.CONF`) that can be turned on and off for individual systems.

Choose "Yes" in response to the following question if you want to enter new command-line arguments:

```
You can specify optional command-line arguments for the server
below. (For example, specify "-D<name>" to define a name for the
<IfDefine> directives or specify "-d<path>" to specify the
ServerRoot directory.) Note that the optional arguments are case-
sensitive.

There are currently no optional command-line arguments.
```

```
Change this value? [NO] Yes
```

Then enter the command-line argument(s) when prompted, as in the following example:

Setting a command-line argument:

```
New command-line arguments: -DSample
```

Removing the argument by leaving the optional argument blank (a null string):

```
Current arguments: "-DSample"

Change this value [NO] Yes
New command-line arguments:
```

Verifying an SSL connection

The server now has a self-signed server certificate, meaning that clients can establish secure (encrypted) connections with your server.

Note: For purposes of a production environment, your server certificate should normally be signed by a third-party commercial certificate authority.

To verify that your SSL-aware server is working:

1. Start your server in the normal way:

   ```
   $ @SYS$STARTUP:APACHE$STARTUP.COM
   ```

2. Connect to it from a client browser by appending "s" to "http" in the URL:

   ```
   http<b>s</b>://<my_server>
   ```

In Netscape Navigator you should see the **New Site Certificate** wizard, and in Internet Explorer you should see the **Security Alert** dialog. As a client, you can choose between not proceeding or proceeding with or without permanently installing the server certificate as a "trusted root certificate authority."

Disabling SSL

You can disable SSL on *SWS* by running the configuration utility. Customizations you have made to your mod_ssl directives and certificates you have generated with the OpenSSL Certificate Tool are preserved.

1. Run the configuration utility:

   ```
   $ @SYS$MANAGER:APACHE$CONFIG.COM
   ```

   Choose "No" in response to the question:

   ```
   Do you want to enable the security features provided by MOD_SSL?
   If so, the server will support the HTTPS (HTTP over the Secure
   Socket Layer) protocol.

   Enable MOD_SSL? [YES] No
   ```

2. Restart the server (confirming the APACHE$WWW processes have stopped):

   ```
   $ @SYS$STARTUP:APACHE$STARTUP.COM

   $ SHOW SYSTEM/PROC=APACHE*

   $ @SYS$STARTUP:APACHE$SHUTDOWN.COM
   ```

Introduction to SSL

What is SSL?

Secure Sockets Layer (SSL) is the open standard security protocol for the secure transfer of sensitive information over the Internet. Implementing SSL requires software to be installed in servers and on browsers that use the SSL protocol. SSL provides three things: privacy through encryption, server authentication, and message integrity. Client authentication is available as an optional function.

With your SSL-aware *HP Secure Web Server* you can ensure a level of security that cannot be achieved by other means. SSL is the most widely used secure method for transmitting sensitive information across the Internet, extranets, and intranets.

> With the growth of the Internet and digital data transmission, many applications need to securely transmit data to remote applications and computers. SSL was originally developed by Netscape to solve this problem using a server-independent architecture. In point-to-point connections, SSL enables mutual authentication between servers and clients by establishing an authenticated and encrypted connection.
>
> SSL runs above TCP/IP and below HTTP, LDAP, IMAP, NNTP, and other high-level network protocols. It provides protection against eavesdropping, tampering, and forgery. Clients and servers are able to authenticate each other and to establish a secure link, or "pipe," across the Internet or intranets to protect the information transmitted.

Important: SSL data transport requires encryption. Many governments, including the United States, have restrictions on the import and export of cryptographic algorithms. Please ensure that your use of SSL is in compliance with all national and international laws that apply to you.

How widely used is SSL?

SSL is a cooperative technology, requiring reciprocating server and client technologies. Both Netscape and Microsoft have built full-featured SSL security into their browsers.

Security and trust are pivotal to the rapid development of eBusiness. More and more web sites are using the SSL protocol to offer clients secure connections and to exchange confidential information. In addition to server-side security, client authentication, also using the SSL protocol for digital IDs and signatures, is gaining much wider acceptance.

By convention, Web pages that require an SSL connection start with https: instead of http: (in the browser's address field). Whenever you enter a secure connection, your browser also shows the familiar padlock image in the status bar, indicating that the page is encrypted.

SSL security symbols in Netscape Navigator and Microsoft Internet Explorer status bars

Depending on your browser and its security settings, you may be unaware of the authentication process unless you are prompted to install a certificate issued by the server. This is because your browser has a store of certificates signed by the same certifying authorities as most servers use (such as VeriSign, for example). You can easily view your certificate store and the details of individual certificates.

SSL is not Secure HTTP

Another protocol for transmitting data securely over the World Wide Web is Secure HTTP (S-HTTP). Encryption of the transport layer allows SSL to be application-independent, while S-HTTP is limited to the specific software implementing it. Both protocols have been approved by the Internet Engineering Task Force (IETF) as a standard.

How are Apache-SSL, mod_ssl, and OpenSSL related?

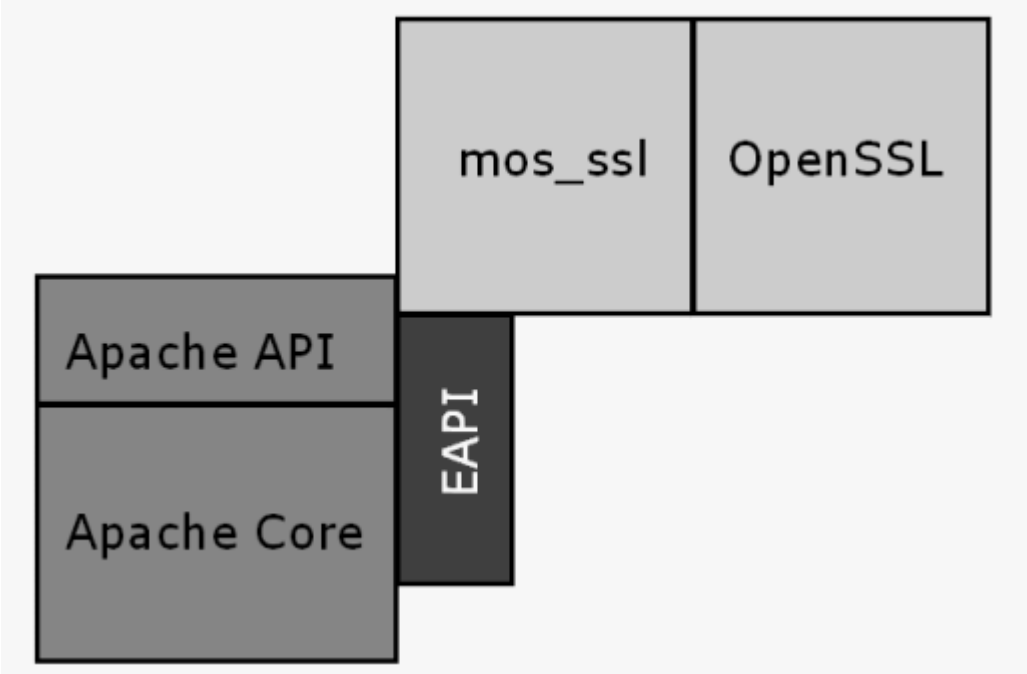
Fortunately, open-source implementations of SSL for Apache are available. The original Apache implementation of SSL was **Apache-SSL**. Subsequently, **mod_ssl** was derived from Apache-SSL and has become an alternative to it. In open source terminology, mod_ssl is a "split" - derived from Apache-SSL but extensively redeveloped, so the code now bears little relation to the original.

Apache-SSL continues to be developed and maintained, with the focus being on on reliability, security and performance within a limited feature set. The increasing popularity of mod_ssl among Apache users is a result of its added-value features and quality. The mod_ssl package is not standalone: it works in conjunction with OpenSSL. The OpenSSL packaged contained in *HP Secure Web Server* uses RSA Security Data's BSAFE cryptographic library.

OpenSSL represents a collaborative effort to develop a robust, commercial-grade, full-featured, and open-source toolkit. It implements the SSL Version 2 and 3 and **Transport Layer Security (TLS)** Version 1 protocols, as well as a full-strength, general-purpose cryptography library.

How does mod_ssl fit into *HP Secure Web Server*?

You can think of mod_ssl as the glue joining OpenSSL with *HP Secure Web Server*. The mod_ssl interface provides Apache web server (on which *SWS* is based) with full use of the OpenSSL toolkit.

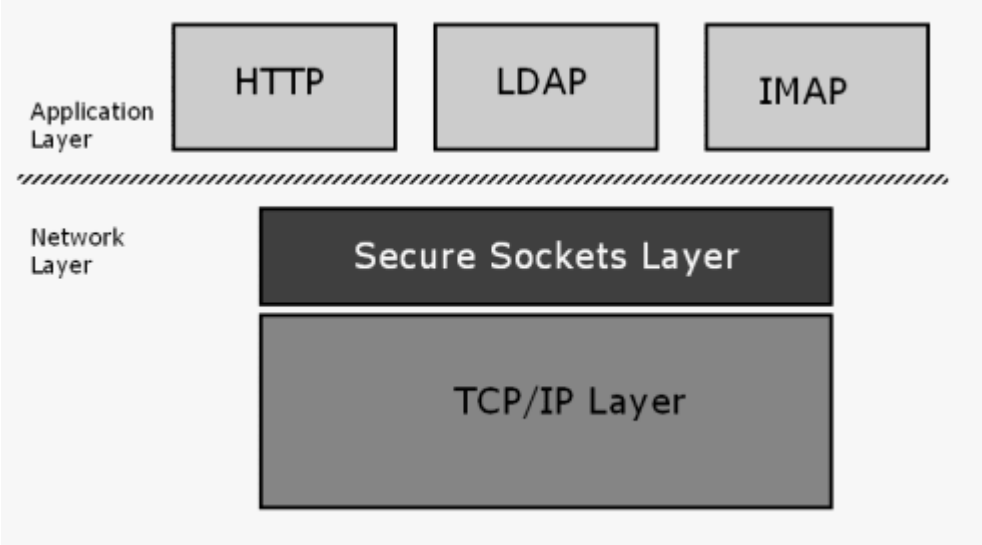


The mod_ssl package integrates the OpenSSL module with a set of source patches for Apache called the **Extended API (EAPI)**. These components are included and automatically installed in *HP Secure Web Server*: the OpenVMS implementation of Apache with mod_ssl.

An SSL Primer

The SSL Protocol

The SSL Protocol works cooperatively with several other protocols. The underlying mechanism is TCP/IP (Transmission Control Protocol/Internet Protocol), which governs the transport and routing of data over the Internet. Application protocols, such as HTTP (HyperText Transport Protocol), LDAP (Lightweight Directory Access Protocol), and IMAP (Internet Messaging Access Protocol), run above TCP/IP. They use TCP/IP to support typical application tasks such as displaying web pages or running email servers.



The SSL protocol runs above TCP/IP and below HTTP, LDAP, and IMAP.
It uses TCP/IP on behalf of these high-level protocols.

SSL addresses three fundamental security concerns about communication over the Internet and other TCP/IP networks:

- **SSL server authentication** allows a user to confirm a server's identity. SSL-enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. For example, if a PC user is sending a credit card number to make a purchase on the web and wants to check the receiving server's identity.
- **SSL client authentication** allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority listed in the server's list of trusted CAs. For example, if a bank sending confidential financial information to a customer and wants to check the recipient's identity.
- **An encrypted SSL connection** requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software, thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering - that is, for automatically determining whether the data has been altered in transit.

The SSL Handshake

An SSL session always begins with an exchange of messages called the **SSL handshake.** The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of **symmetric keys**, which are used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server.

This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.
- Establish an encrypted SSL connection.

What is public key encryption?

In traditional, non-Internet environments, encrypted information is sent between parties that both use the same key to encode and decode information. This is called **symmetrical encryption**. In the case of the Internet, there is no way for one computer to send the encryption key to another without the risk that a third party can steal the key and decode subsequent communications. A method other than symmetrical encryption is required to transmit the encryption key securely on the Internet.

> The principals of public key cryptography were developed by Whitfield Diffie and Martin Hellman. The **Diffie-Hellman** key agreement protocol was published in 1976. It is also called **asymmetric** encryption because it uses two keys instead of one key.

The solution is a system which uses two keys. The first is a **public key**, and usually available to anybody who wants it. The second, a **private key**, is held by just one party. Only the private key can decipher information encrypted using the public key; it is impossible to decipher the message using the public key. Similarly, only the private key can create encrypted messages decipherable with the public key. Because there can be only one public key for each private key, and vice-versa, it is nearly impossible for anybody to impersonate the holder of the private key. The two keys are mathematically related, but in such a way that it is virtually impossible for anybody to derive the private key from the public one.

During the SSL handshake, each computer generates a set of codes to encrypt information. From these codes, each computer creates two **keys**, one private and one public. Your computer keeps the private key secret, but sends out the public key to the other computer, which uses that key to encode subsequent messages so that only your computer can read them. The public key cannot, however, be used to decode the message; the decoding can only be done using the private key.

These keys allow you and the other computer to lock and unlock information so that only the holder of the private key can read messages encrypted by the public key. Since only you and the other computer have a copy of your respective private keys, there is no way for anybody else to intercept and decode your messages.

The secure link

When a user enters a secure link to send information with either Netscape Communicator or

Microsoft Internet Explorer, the browser negotiates the key code exchanges so the user is not aware of this happening. However, the page information downloads more slowly on the secure link than it does on unsecured links because of the extra encryption information being sent. Both the user's computer and the server computer generate a public-private key set, and then exchange public keys with each other.

Once this exchange has occured, a new master key is generated and transmitted through the secure connection. This master key is symmetrical; messages can now be both encrypted and decrypted using the same key. In addition, a message authentication code (MAC) is used to make sure that the information being exchanged is not altered during transmission.

How do certificates work?

A **certificate**, or **digital certificate**, is an electronic document used to identify an individual, a server, a company, or some other entity and to associate that identity with a public key. Like a driver's license, a passport, or other commonly used personal IDs, a certificate provides generally recognized proof of a person's identity. Public-key cryptography uses certificates to address the problem of impersonation.

Certificates are issued by **Certificate authorities (**also known as **Certification authorities)**. These are trusted third parties that verify the identity of the site you are connected with. Like any form of identification, the authenticity of the issuer is essential.

> The role of CAs in validating identities and issuing certificates is analogous to the way a government issues passports and driver's licenses. CAs can be either independent third parties or organizations running their own certificate-issuing server software (such as Netscape Certificate Server).

The methods used to validate an identity vary depending on the policies of a given CA. In general, before issuing a certificate, the CA must use its published verification procedures for that type of certificate to ensure that an entity requesting a certificate is in fact who it claims to be.

The certificate issued by the CA binds a particular public key to the name of the entity the certificate identifies (such as the name of an employee or a server). Certificates help prevent the use of fake public keys for impersonation. Only the public key certified by the certificate will work with the corresponding private key possessed by the entity identified by the certificate.

In addition to a public key, a certificate always includes the name of the entity it identifies, an expiration date, the name of the CA that issued the certificate, a serial number, and other information. Most importantly, a certificate always includes the **digital signature** of the issuing CA. The CA's digital signature allows the certificate to function as a "letter of introduction" for users who know and trust the CA but don't know the entity identified by the certificate.

How to view browser certificates

Netscape and Microsoft browsers have built-in lists of CAs, which you can alter if you want to. There is no absolute list of which Certificate Authorities are reliable, but the ones included in Netscape and Microsoft browsers have been accepted as dependable by Netscape and Microsoft. If you connect with a secure site authorized by a CA not listed in your browser's list, you will be alerted and asked if you want to add the new CA to your browser's list. It is not recommended that you add a new CA to your list, unless you have a good reason to trust the CA.

Viewing certificates in Microsoft Internet Explorer
To view CA certificates that are contained in the browser's Certificate Manager:

1. On the **Tools** menu in Internet Explorer, click **Internet Options**.

2. Click the **Content** tab.
3. In the **Certificates**area, click the **Certificates** button to view the list of current certificates by category, including Trusted Root Certification Authorities.

Viewing certificates in Netscape Navigator
To view the CA certificates that are contained in the browser's Trusted Root Library:

1. On the Netscape toolbar, click the Security icon to open the **Security Info** window.
2. Click the link labeled **Signers** to open the **Certificate Signers' Certificates** window, containing a list of all the CA certificates contained in the browser.

In addition, you can view the specific certificate being used in a secure connection by double-clicking on the padlock symbol in your browser's status bar.

How does SSL use ciphers?

Integral to the SSL protocol is its use of cryptographic algorithms, generally called **ciphers.** These are required to authenticate the server and client to each other, transmit certificates, and establishing session keys. Clients and servers may support different **cipher suites,** or sets of ciphers, depending on factors such as the version of SSL they support, company policies regarding acceptable encryption strength, and government restrictions on export of SSL-enabled software.

Among its other functions, the SSL handshake protocol determines how the server and client negotiate which cipher suites they will use to authenticate each other, to transmit certificates, and to establish session keys. Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they will both use during an SSL session. The most commonly used SSL cipher suites use RSA key exchange.

The SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session.

Decisions about which cipher suites a particular organization decides to enable depend on trade-offs among the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

How do digital signatures work?

Encryption and decryption address the problem of eavesdropping. However, tampering and impersonation are still possible.

Public-key cryptography addresses the problem of tampering using a mathematical function called a **one-way hash** (also called a **message digest**). A one-way hash is a fixed-length number who value is unique to the data being hashed. Any change in the data, even deleting or altering a single character, results in a different value.

The content of the hashed data cannot, for all practical purposes, be deduced from the hash, which is why it is called "one-way."

This principal is the crucial part of digitally signing any data. Instead of encrypting the data itself, the signing software creates a one-way hash of the data, then uses your private key to encrypt the hash. The encrypted hash, along with other information, such as the hashing algorithm, is known as a **digital signature**.

What are certificate chains?

The X.509 standard (the certificate protocol used by SSL) includes a model for setting up a hierarchy of CAs, making it possible to delegate certificate-issuing responsibilities to subordinate CAs. Inspecting a browser's certificate store will show a collection of "intermediate CAs."

CA hierarchies are reflected in **certificate chains**. A certificate chain is a succession of certificates issued by successive CAs. Trusted **root CAs** are at the pinnacle of the pyramid and are the only entities to self-sign their certificates.

Using the mod_ssl directive **SSLVerifyDepth** you can determine how many levels of intermediate CAs you would like your server to authenticate.

Using mod_ssl directives

The **mod_ssl** directives are your means for configuring **OpenSSL** to function in exactly the way you want for your SSL-enabled *HP Secure Web Server*. All mod_ssl directives can be applied to the main server configuration file (HTTPD.CONF) by inclusion in the MOD_SSL.CONF include file.

How to apply mod_ssl directives

There are three classes of mod_ssl directives used by *HP Secure Web Server*:

- **Global Directives** Although you can put these anywhere in the HTTPD.CONF file, you should use the MOD_SSL.CONF include file outside any sectioning commands like <VirtualHost>.
- **Per Server Directives** Use the MOD_SSL.CONF include file, either outside sections (for the main/default server) or inside <VirtualHost> sections.
- **Per Directory Directives** Use the the MOD_SSL.CONF include or the per-directory .HTACCESS files.

The three classes of directives are hierarchical: per directory directives can also be used in the per server and global context. Per Server directives can also be used in the global context.

Entering directives in the server configuration file

Directives in the MOD_SSL.CONF file are included in the HTTPD.CONF server configuration file. Like any change to HTTPD.CONF, it has no effect until shutting down and restarting the server.

Since MOD_SSL.CONF is an include file, changes to it are not affected by disabling SSL. When you reenable it, the same file is included in HTTPD.CONF again.

Note: Although mod_ssl permits many directives to be entered in other configuration files, you should not add mod_ssl directives directly to HTTPD.CONF or to other configuration files (including SRM.CONF and ACCESS.CONF).

To edit the MOD_SSL.CONF file, use this command:

```
$ EDIT APACHE$COMMON:[CONF]MOD_SSL.CONF
```

if you are using common configuration files across a cluster

or

```
$ EDIT APACHE$SPECIFIC:[CONF]MOD_SSL.CONF
```

if you are using system-specific configuration files

Warning: Editing MOD_SSL.CONF or HTTPD.CONF can affect or prevent your server from running. If necessary, copies of these file exist:

```
APACHE$COMMON:[APACHE.SRC.OS.OPENVMS]MOD_SSL.CONF
APACHE$COMMON:[APACHE.SRC.OS.OPENVMS]HTTPD.CONF-DIST-OPENVMS-SSL
```

Using access files

Using access files (by default .HTACCESS files) is more flexible, but puts a greater burden on performance and security. Remember also that the default setting in HTTPD.CONF is **AllowOverride None**, meaning directives in .HTACCESS files are ignored. Overrides are activated by the **AllowOverride** directive, and apply to a particular scope (such as a directory) and all descendants, unless further modified by other **AllowOverride** directives at lower levels.

Here are some usage guidelines:

- *HP Secure Web Server* automatically looks for the access files in each document directory.
- You do not have to restart the server after changing an access file's contents.
- You can redefine the name of the access files (.HTACCESS by default) with the **AccessFileName** directive in HTTPD.CONF (using the MOD_SSL.CONF include file).
- The contents of the access files are treated as if they are in the <Directory> section of MOD_SSL.CONF. Therefore, you should not use the <Directory> sectioning command inside the access files.

Mapping mod_ssl contexts

The *www.modssl.org* documentation refers to contexts such as **server config** and **virtual host**. Different directives may be applied in different contexts, and these determine the scope of their effect. These contexts should be understood as follows:

server config
This context means that you can use the directive in HTTPD.CONF (using the MOD_SSL.CONF include file) but not within any <VirtualHost> or <Directory> containers. It is not allowed in .HTACCESS files at all.

virtual host
This context means that you can use the directive in HTTPD.CONF (using the MOD_SSL.CONF include file) but only inside <VirtualHost> sections of HTTPD.CONF.

directory
The <Directory> section of MOD_SSL.CONF should specify the same path as the **DocumentRoot** does. By default:

```
<Directory "/apache$common/htdocs">
```

Each directory to which *SWS* has access can be separately configured with respect to which services and features are allowed and/or disabled in that directory (and its subdirectories).

location
By default, all requests are taken from the **DocumentRoot** directory, but you can use symbolic links and aliases to point to other locations. For example:

```
<Location /server-info>
```

Summary of mod_ssl directives

In the following summary listing, HTTPD.CONF (using the MOD_SSL.CONF include file) is used where "`server config`" appears in the mod_ssl official documentation.

Use these directives to determine how the SSL Engine will operate:

SSLEngine

Description:	Switches the SSL Engine on or off.
Syntax:	SSLEngine *on*|*off*
Default:	SSLEngine on
Context:	HTTPD.CONF, virtual host

SSLProtocol

Description:	Configures usable SSL protocol flavors.
Syntax:	SSLProtocol [+-]protocol where [+-]*protocol* can be SSLv2, SSLv3, TLSv1, or All.
Default:	SSLProtocol all
Context:	HTTPD.CONF, virtual host
Override:	Options

SSLLog

Description	Specifies where to write the dedicated SSL engine logfile.
Syntax:	SSLLog *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLLogLevel

Description:	Sets the logging level for the dedicated SSL engine logfile.
Syntax:	SSLLogLevel *level* *where level can be* none*,* error*,* warn*,* info*,* trace*, and* debug
Default:	SSLLogLevel none
Context:	HTTPD.CONF, virtual host
OpenVMS usage:	

Use these directives to set server startup and administration:

SSLPassPhraseDialog

Description:	Determines the type of pass-phrase dialog for decrypting private keys at startup time. The default requires manual entry of pass phrases.
Syntax:	SSLPassPhraseDialog *type* *where type is builtin or* exec:/path/to/program
Default:	SSLPassPhraseDialog builtin
Context:	HTTPD.CONF
OpenVMS note:	Do not use an encoded pass phrase with the builtin option.

SSLMutex

Description:	Provides a method for mutual exclusion of internal operations.
Syntax:	SSLMutex *type*

	where type is none, file:/path/to/mutex, sem, *or* csem
Default:	SSLMutex none
Context:	HTTPD.CONF
OpenVMS note:	OpenVMS uses semaphore-caching mutex, because it's faster than file locking.

SSLRandomSeed

Description:	Configures one or more sources for seeding the Pseudo Random Number Generator (PRNG) in OpenSSL at startup time.
Syntax:	SSLRandomSeed *context source* [*bytes*] *where context source is* builtin, file:/path/to/source, *or* exec:/path/to/program.
Default:	none
Context:	HTTPD.CONF

Use these directives to determine how a secure connection should be established and maintained with the client:

SSLCipherSuite

Description:	Specifies the cipher suite for negotiation in the SSL handshake.
Syntax:	SSLCipherSuite *cipher-spec*
Default:	SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
Context:	HTTPD.CONF, virtual host, directory, .HTACCESS
Override:	AuthConfig

SSLSessionCache

Description:	Configures storage type of the global/interprocess SSL Session Cache.
Syntax	SSLSessionCache type where type is none, dbm:/path/to/datafile, SHM, *or* CSHM
Context:	HTTPD.CONF
OpenVMS note:	OpenVMS uses a file-based session cache. (OpenVMS does not support a shared-memory session cache at this time.)

SSLSessionCacheTimeout

Description:	Sets the number of seconds before an SSL session expires in the Session Cache.
Syntax:	SSLSessionCacheTimeout *seconds*
Default:	SSLSessionCacheTimeout 300
Context:	HTTPD.CONF, virtual host

Use these directives to specify the file or directory locations of certificate, key, chain, and revocation files:

SSLCertificateFile

Description:	Specifies the server PEM-encoded X.509 Certificate file.
Syntax:	SSLCertificateFile *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCertificateKeyFile

Description:	Specifies the server PEM-encoded Private Key file.
Syntax:	SSLCertificateKeyFile *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCertificateChainFile

Description:	Specifies a file with concatenated PEM-encoded server CA certificates.
Syntax:	SSLCertificateChainFile *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCACertificatePath

Description:	Specifes the directory of PEM-encoded CA certificates for client authorization.
Syntax:	SSLCACertificatePath *directory*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCACertificateFile

Description:	File of concatenated PEM-encoded CA certificates for client authorization.
Syntax:	SSLCACertificateFile *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCARevocationPath

Description:	Directory of PEM-encoded CA client revocation lists for client

	authorization.
Syntax:	SSLCARevocationPath *directory*
Default:	*None*
Context:	HTTPD.CONF, virtual host

SSLCARevocationFile

Description:	File of concatenated PEM-encoded CA client revocation lists for client authorization.
Syntax:	SSLCARevocationFile *filename*
Default:	*None*
Context:	HTTPD.CONF, virtual host

Use these directives to enforce secure connections according to the level of server and client authentication you want:

SSLVerifyClient

Description:	Specifies the type of Client Certificate verification.
Syntax:	SSLVerifyClient *level*
Default:	SSLVerifyClient none
Context:	HTTPD.CONF, virtual host, directory, .HTACCESS

SSLVerifyDepth

Description:	Sets the maximum depth of CA certificates in client certificate verification.
Syntax:	SSLVerifyDepth *number*
Default:	SSLVerifyDepth 1
Context:	HTTPD.CONF, virtual host, directory, .HTACCESS
Override:	AuthConfig

SSLRequireSSL

Description:	Denies client access when not using an https request.
Syntax:	SSLRequireSSL
Default:	*None*
Context:	directory, .HTACCESS
Override:	AuthConfig

SSLRequire

Description:	Allows client access only when a custom Boolean expression is True.

Syntax:	SSLRequire *expression*
Default:	*None*
Context:	directory, .htaccess
Override:	AuthConfig

SSLOptions

Description:	Configures various SSL engine run-time options
Syntax:	SSLOptions [+-]*option* ...
	where *option* can be StdEnvVars, CompatEnvVars, ExportCertData, FakeBasicAuth, StrictRequire, and OptRenegotiate.
Default:	*None*
Context:	HTTPD.CONF, virtual host, directory, .HTACCESS
Override:	Options

Understanding Certificates

This chapter explains the fundamentals of certificate contents. The next chapter shows you how to use *HP Secure Web Server*'s **OpenSSL Certificate Tool**, a simple interface for working with certificates. The final chapter gives you the how-to information you'll need to put certificates in action on your server and in your organization.

The anatomy of a certificate

SSL certificates can be used to authenticate servers or clients. The contents of most certificates are organized according to the X.509 V3 certificate specification, as recommended by the International Telecommunications Union (ITU)

Distinguished names

A digital certificate binds a **distinguished name (DN)** to a **public key**.

Distinguished names provide an identity in a specific context. Distinguished names are defined by the X.509 standard [X509], which defines the fields, field names, and abbreviations used to refer to the fields.

A DN is actually a series of names that uniquely identifies the certificate **subject.** The subject of a server certificate is identified by country, state, city, organization, unit, and server name.

DNs may include a variety of other name-value pairs. They are used to identify both certificate subjects and entries in directories that support LDAP (Lightweight Directory Access Protocol).

Distinguished Name Field	Abbreviation	Description	Example
Country	C	Name is located in this Country (ISO code)	US
State/Province	ST	Name is located in this State/Province	Illinois
City/Locality	L	Name is located in this City	Metropolis
Organization or Company	O	Name is associated with this organization	XYZ Corp.
Organizational Unit	OU	Name is associated with this organization unit, such as a department	Research Dept.
Common Name	CN	Name being certified	TEST.RES.XYZ.COM

A typical certificate

Every X.509 certificate consists of two sections:

The data section includes the following information:

- The **version number** of the X.509 standard supported by the certificate.
- The certificate's **serial number**. Every certificate issued by a CA has a serial number that is unique to the certificates issued by that CA.

- Information about the user's **public key**, including the algorithm used and a representation of the key itself.
- The **DN** of the CA that issued the certificate.
- The period during which the certificate is valid (for example, between 1:00 p.m. on January 1, 2004 and 1:00 p.m. December 31, 2004)
- The DN of the certificate subject (for example, in a client SSL certificate this would be the user's DN), also called the subject name.
- Optional **certificate extensions,** which may provide additional data used by the client or server. For example, the certificate type extension indicates the type of certificate - that is, whether it is a client SSL certificate, a server SSL certificate, a certificate for signing email, and so on. Certificate extensions can also be used for a variety of other purposes.

The signature section includes the following information:

- The **cryptographic algorithm**, or cipher, used by the issuing **certificate authority (CA)** to create its own digital signature.
- The CA's **digital signature**, obtained by hashing all of the data in the certificate together and encrypting it with the CA's private key.

Types of certificates

Working with SSL certificates in a web server environment involves three types of certificates.

Server certificates

These identify servers to clients via SSL-based server authentication. You can use server authentication with or without client authentication. However, server authentication is a requirement for an encrypted SSL session.

> **Example:** E-commerce sites usually support certificate-based server authentication to encrypt personal information, so that credit card numbers, for example, cannot easily be intercepted.
>
> **With *SWS*'s Certificate Tool:** You can create a certificate request (Option 3) and then self-sign (Option 4) it. Or, in a production environment, you have it signed by a trusted certificiate authority.

Client certificates

These identify clients to servers using SSL-based client authentication. Typically, the identity of the client is assumed to be the same as the identity of a human being, such as an employee in an enterprise.

> **Example:** A corporate intranet might give a new employee a client SSL certificate that allows the company's servers to identify that employee and authorize access to the company's servers.
>
> **With *SWS*'s Certificate Tool:** You can create a client certificate request (using the same option as for a server certificate request) and then sign the request (Option 6) using your own CA certificate.

CA certificates

These identify certificate authorities. They can be trusted root or intermediate certificates that client browser and web servers use CA certificates to determine what other certificates can be trusted.

> **Example:** The CA certificates stored in your web browser (either Internet Explorer or

Netscape Navigator) determine what other certificates that browser can authenticate without warning the user that a site has an untrusted certificate.

With *SWS*'s Certificate Tool: You can create a certificate authority (CA) certificate using Option 5.

Using the Certificate Tool

HP Secure Web Server provides a simple interface for viewing and creating SSL certificates. The **OpenSSL Certificate Tool** enables you to perform the most important certification functions with ease. Using it, you can view certificates and certificate requests, create certificate requests, sign your own certificate, create your own certificate authority, and sign client certificate requests. Additional hash functions are included.

Note: Some OpenSSL commands are beyond the scope of the Certificate Tool. For these, you'll need to use command-line OpenSSL.

Start the tool

Run the Certificate Tool with the following command:

```
$ @APACHE$COMMON:APACHE$CERT_TOOL.COM
```

View a certificate file

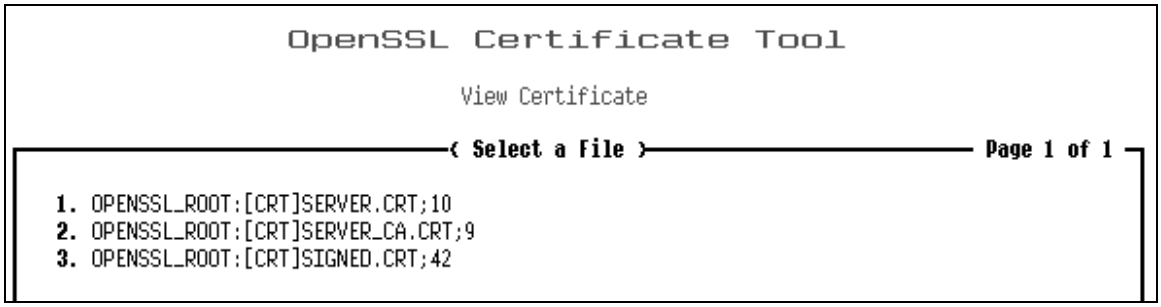
The contents of a certificate associate a **public key** with the real identity of an individual, server, or other entity, known as the **subject**. Information about the subject includes identifying information (the **distinguished name**), and the public key. It also includes the identification and signature of the **Certificate Authority** that issued the certificate, and the period of time during which the certificate is valid. It may have additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Do the following:

1. Accept the default file specification (or type a new to an alternate location] to the certificate directory to find files with a CRT extension:

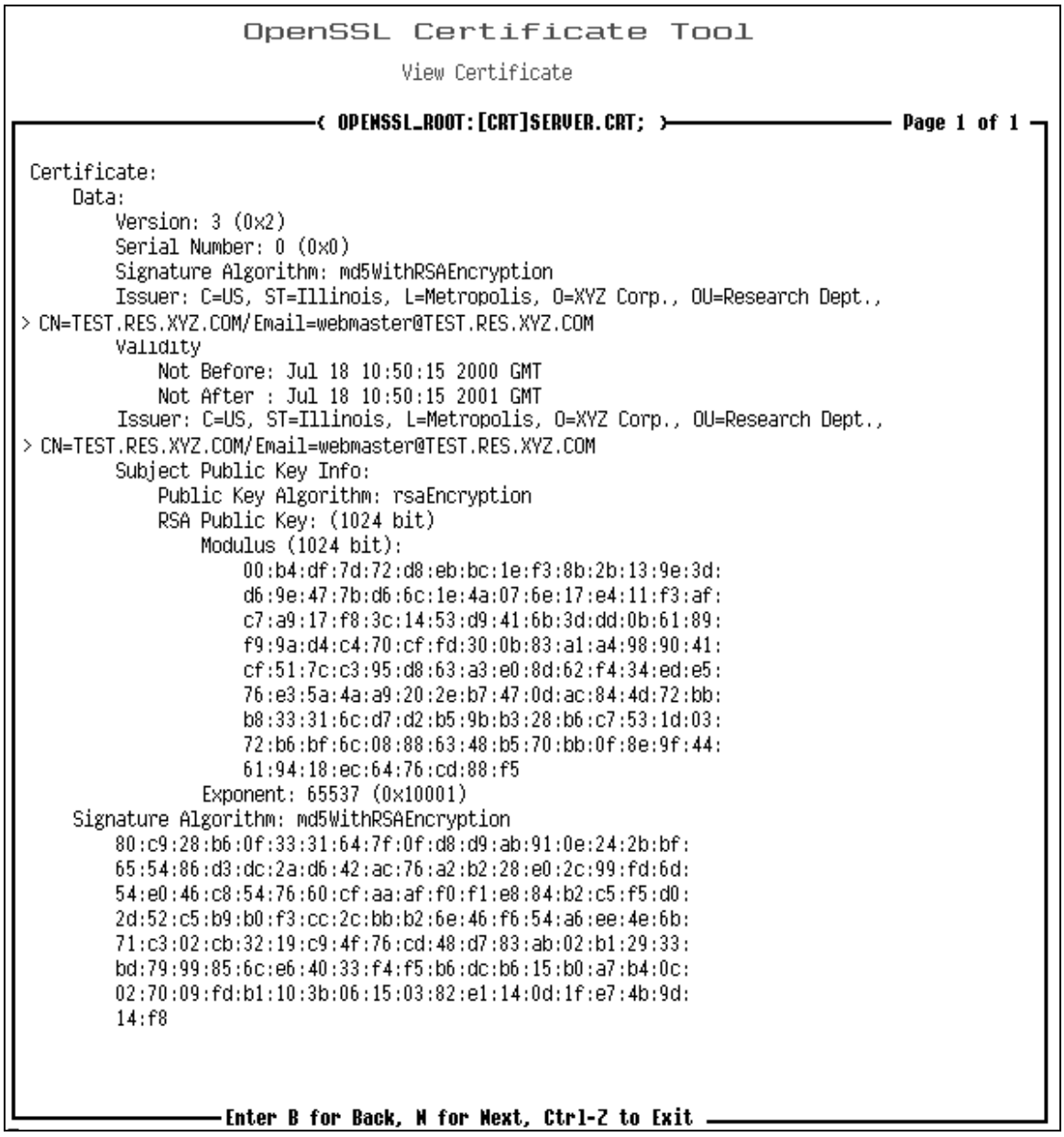
The default file specification `OPENSSL_ROOT:[CRT]` is where certificates you sign are saved. Server certificates installed on your system can be found in `APACHE$COMMON:[CONF.SSL_CRT]` or `APACHE$SPECIFIC:[CONF.SSL_CRT]`.

2. Select a certificate file:



3. View the certificate details:

-Version SSL 3.0 protocol
-Serial number Certificates issued by a CA have a serial number that is unique to the certificates issued by that CA.
-Signature Algorithm
-Issuer
-Validity (inception and expiration dates)
-Public key information



View a certificate request file

A certificate request file is an unsigned certificate. It can be a server certificate request or a client certificate request.

Do the following:

1. Type the file specification to the certificate request directory to find files with a CSR extension:

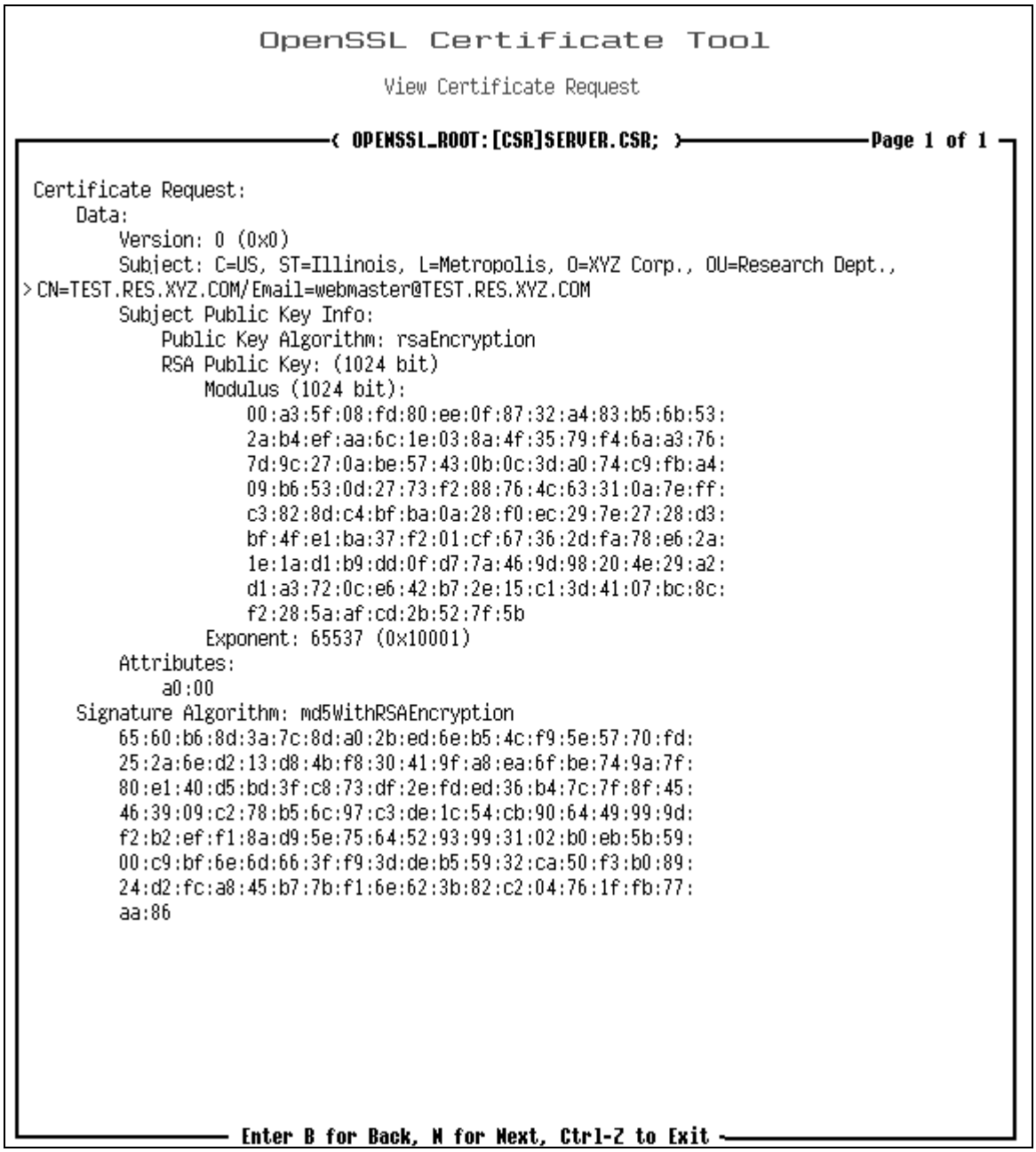
2. Select a certificate request file:

3. View the certificate request details:

-Subject

-Public key information

-Signature Algorithm
-Issuer
-Validity (inception and expiration dates)



Create a certificate request

You can think of creating a certificate request (generating a *.CSR file) as representing an application form for a certificate. There are two categories of request:

Server certificate request
This means preparing a certificate file to be signed by a trusted (root) CA in order to authenticate your server. You are the **subject** of the certificate and the CA you send it to will be the certificate **issuer**. For example, if you wanted to get a Thawte Server ID, you would create a certificate request and email the contents of this generated file to Thawte. The file you generate is a *.CSR file.

Client certificate request
This means preparing client certificate files that you sign and distribute to clients in order to authenticate them. The client is the **subject** of the certificate and you are the certificate **issuer**.

Do the following:

1. Enter the required information for the certificate:

-**Encrypt Private Key?** Using an encrypted private key forces the pass-phrase dialog to appear at startup time, requiring manual input.

Usage note: Do not use this option if using the mod_ssl directive **SSLPassPhraseDialog** with the default **builtin** option.

-Encryption Bits? 1024 bits is the largest recommended size.

Explanation: Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Key length is measured in bits. Private key sizes larger than 1024 bits are incompatible with some versions of Netscape Navigator and Microsoft Internet Explorer.

-Certificate Key File? Use OpenVMS syntax (usually, `[OPENSSL_ROOT:[KEY]SERVER.KEY]`)
-Certificate Request File? Use OpenVMS syntax (usually, `[OPENSSL_ROOT:[CRT]SERVER.CRT]`)

-Country Name? The remaining questions determine your server's **Distinguished Name**

-State or Province Name?
-City Name?
-Organization Name?
-Organization Unit Name?

-Common Name? Common name usage is different for client certificates than it is for server certificates. The common name on a client certificate is generally the proper name of the individual requesting a certificate. In the case of server certificates, the common name must be the same as your server's DNS host name (or virtual host name, if name-based virtual hosting is used).

Explanation: Browsers compare the common name in the server certificate with the host name of the server they are connecting to. These must match.

-Email Address?
-Display the Certificate?

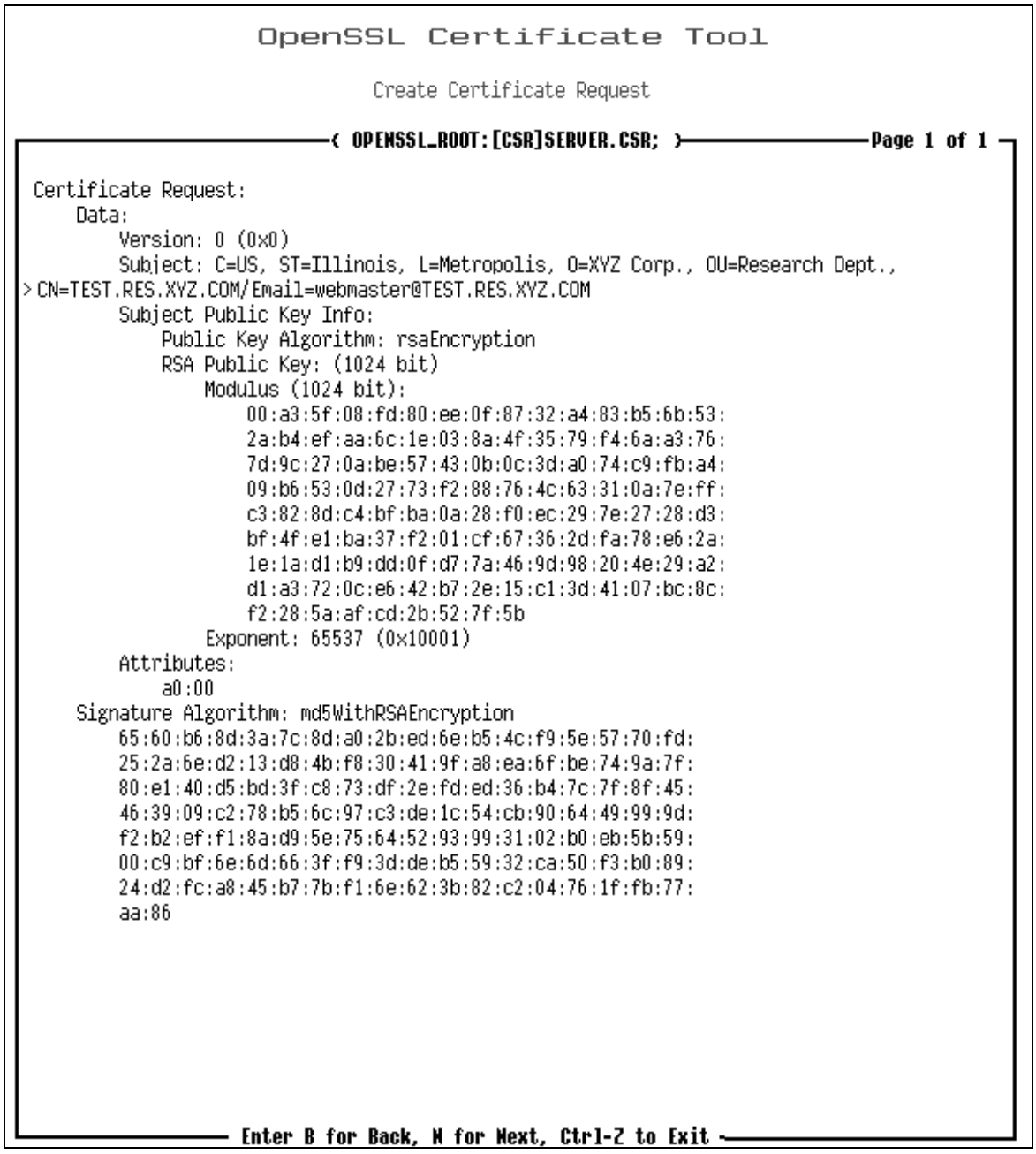
Important: All fields must be completed to create a valid certificate request.

```
                    OpenSSL Certificate Tool

                        Create Certificate Request

Encrypt Private Key ? [N]
Encryption Bits ? [1024]
Certificate Key File ? [OPENSSL_ROOT:[KEY]SERVER.KEY]
Certificate Request File ? [OPENSSL_ROOT:[CSR]SERVER.CSR]
Country Name ? [US]
State or Province Name ? [Illinois]
City Name ? [Metropolis]
Organization Name ? [XYZ Corp.]
Organization Unit Name ? [Research Dept]
Common Name ? [ TEST.RES.XYZ.COM ]
Email Address ? [webmaster@TEST.RES.XYZ.COM]
Display the Certificate ? [N] Y
```

The certificate request is generated after responding to the last question.

2. View the details of the certificate request (if you chose to display the certificate):

-Subject
-Public key information
-Signature Algorithm



To see the encoded contents, exit the configuration utility and view the CSR file.

```
$ TYPE OPENSSL_ROOT:[CSR]SERVER.CSR
```

What you see is exacly what is required by the Certificate Authority. You may be required to send the file itself or just the contents of the file to your CA (according to the CA's instructions).

For example:

```
-----BEGIN CERTIFICATE REQUEST-----
 MIIB/TCCAWYCAQAwgbwxCzAJBgNVBAYTAlVTMRYwFAYDVQQIEw1OZXcgSGFtcHNo
 aXJlMQ8wDQYDVQQHEwZOYXNodWExHjAcBgNVBAoTFUNvbXBhcSBDb21wdXRlciBD
 b3JwLjEcMBoGA1UECxMTT3BlblZNUyBFbmdpbmVlcmluZzEaMBgGA1UEAxMRRkxJ
 UDMuWktPLkRFQy5DT00xKjAoBgkqhkiG9w0BCQEWG3dlYm1hc3RlckBGTElQMy5a
 S08uREVDLkNPTTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0/y8RxuE/COy
```

```
nVpeK00GgvbgFWxX1o89ULQTMVUSwmAzhdzbi3DZL5s85YRGdPVgYW2rWs1t2SQg
jMSlFTxta/CwW6Vwwn9GmdaJwkqGFxnpw2LmugexLfj+4t97AZyIR2O7gJxCINS5
CWg3tcn1ZUmqswjkrG8WehUN+2C6IBcCAwEAAaAAMA0GCSqGSIb3DQEBBAUAA4GB
ABzgiiojPAcojLXGI2OFxJ5apORAHHHAyc0YCuhFXS1Rs2BIXHmM5xQuxk8yitc4
yViQfHhGDzpDmOwMKkK7t09UjQh9humKEUlAnS4VYLL4VlgenwLybcLLB0Q3aiQN
UjQw9RrXNWWZYVDenvrOwtbK9dFefb4PlZIAS2/Z4jLP
-----END CERTIFICATE REQUEST-----
```

If sending the contents, copy and paste everything and send to the CA using secure email or the appropriate enrollment form. What the CA returns to you will be a digitally signed certificate.

For example:

```
-----BEGIN CERTIFICATE-----
MIICeDCCAiICEEdpjxOzmJPyh5TiG8BRA70wDQYJKoZIhvcNAQEEBQAwgakxFjAU
BgNVBAoTDVZlcmlTaWduLCBJbmMxRzBFBgNVBAsTPnd3dy52ZXJpc2lnbi5jb20v
cmVwb3NpdG9yeS9UZXN0Q1BTIEluY29ycC4gQnkgUmVmLiBMaWFiLiBMVEQuMUYw
RAYDVQQLEz1Gb3IgVmVyaVNpZ24gYXV0aG9yaXplZCB0ZXN0aW5nIG9ubHkuIE5v
IGFzc3VyYW5jZXMgKEMpVlMxOTk3MB4XDTAwMDcwNzAwMDAwMFoXDTAwMDcyMTIz
NTk1OVowgZAxCzAJBgNVBAYTAlVTMRYwFAYDVQQIEw1OZXcgSGFtcHNoaXJlMQ8w
DQYDVQQHFAZOYXNodWExHjAcBgNVBAoUFUNvbXBhcSBDb21wdXRlciBDb3JwLjEc
MBoGA1UECxQTT3BlblZNUyBFbmdpbmVlcmluZzEaMBgGA1UEAxQRRkxJUDMuWktP
LkRFQy5DT00wgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBANP8vEcbhPwjsp1a
XitNBoL24BVsV9aPPVC0EzFVEsJgM4Xc24tw2S+bPOWERnT1YGFtq1rNbdkkIIzE
pRU8bWvwsFulcMJ/RpnWicJKhhcZ6cNi5roHsS34/uLfewGciEdju4CcQiDUuQlo
N7XJ9WVJqrMI5KxvFnoVDftguiAXAgMBAAEwDQYJKoZIhvcNAQEEBQADQQAySLLe
U7nMLJ+QkRld6iqKjU2VotphPvgWMGsJ+TKqUI4MXaAv0zQxtBni1N8s0LXVNCuJ
lEzBYjSbgbgEhJJA
-----END CERTIFICATE-----
```

The CA-signed certificate contains the following:

-Your organization's **common name** (www.<yourserver>)
-Additional identifying information (IP and physical address)
-Your **public key**
-Expiration date of the public key
-Name of the CA that issued the ID
-A unique **serial number.** Every certificate issued by a CA has a serial number that is unique to the certificates issued by that CA.
-CA's **digital signature**

Installing certificates
A signed certificate needs to be installed, along with the key you generated when creating the request, by saving the respective files to their correct directories and restarting the server.

For the certificate file, this is either `APACHE$COMMON:[CONF.SSL_CRT]` or `APACHE$SPECIFIC:[CONF.SSL_CRT]`.

For the key file, this either `APACHE$COMMON:[CONF.SSL_KEY]` or `APACHE$SPECIFIC:[CONF.SSL_KEY]`.

See also

Installing a server certificate

Create a self-signed certificate

Creating a self-signed certificate is an essential first step after installing *SWS* with SSL. The server will not start without the presence of a properly signed and installed certificate. This procedure is performed for you automatically. Therefore, this command is only required if the

certificate file requires changing.

Installing certificates
After signing a certificate, you need to install it by copying the certificate and certificate key to the correct directory and restarting the server. For example:

```
$ COPY APACHE$SPECIFIC:[OPENSSL.CRT]SERVER.CRT
APACHE$SPECIFIC:[CONF.SSL_CRT]

$ COPY APACHE$SPECIFIC:[OPENSSL.KEY]SERVER.KEY
APACHE$SPECIFIC:[CONF.SSL_KEY]
```

Do the following:

1. Enter the required information for the self-signed certificate:

   -**Encrypt Private Key?** Using an encrypted private key forces the Pass Phrase dialog to appear at startup time.

   -Encryption Bits? 1024 bits is the largest recommended size.

   Explanation: Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Key length is measured in bits. Private key sizes larger than 1024 bits are incompatible with some versions of Netscape Navigator and Microsoft Internet Explorer.

   -Certificate Key File? Use OpenVMS syntax (usually, `[OPENSSL_ROOT:[KEY]SERVER.KEY]`)
   -Certificate Request File? Use OpenVMS syntax (usually, `[OPENSSL_ROOT:[CRT]SERVER.CRT]`)

   -Country Name? The remaining questions determine your server's **Distinguished Name**.

   -State or Province Name?
   -City Name?
   -Organization Name?
   -Organization Unit Name?

   -Common Name? This must be the same as your server's DNS host name (or virtual host name, if name-based virtual hosting is used).

   Explanation: Browsers compare the common name in the server certificate with the host name of the server they are connecting to. These must match.

   -Email Address?
   -Display the Certificate?

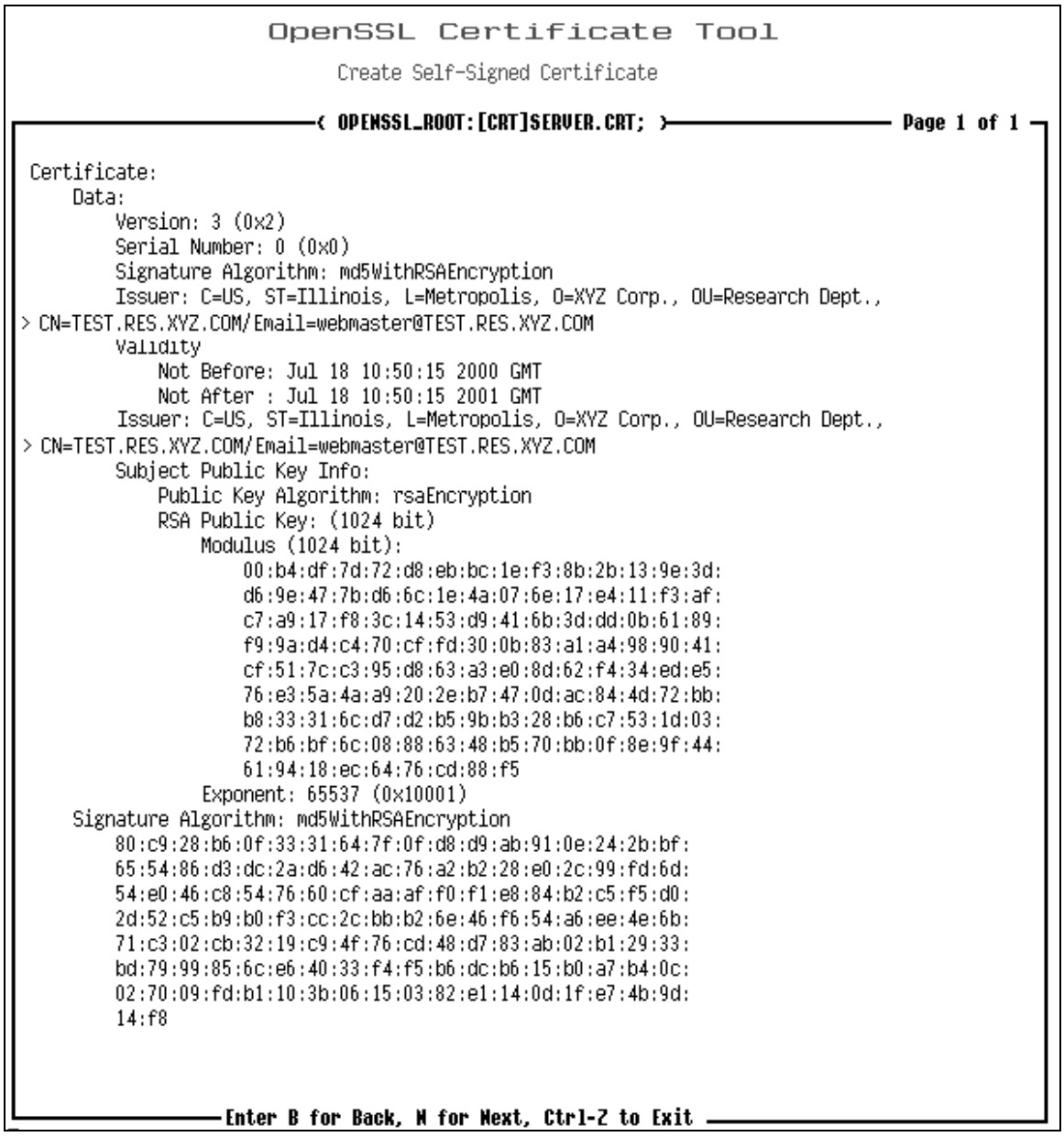
Important: All fields must be completed to create a valid self-signed certificate. The inception time of a certificate is based on UTC (Coordinated Universal Time). Check with your system administrator that your computer's UTC is set correctly if you want to use the self-signed certificate right away.

```
                    OpenSSL Certificate Tool

                        Create Self-Signed Certificate

Encrypt Private Key ? [N]
Encryption Bits ? [1024]
Certificate Key File ? [OPENSSL_ROOT:[KEY]SERVER.KEY]
Certificate Request File ? [OPENSSL_ROOT:[CSR]SERVER.CSR]
Country Name ? [US]
State or Province Name ? [Illinois]
City Name ? [Metropolis]
Organization Name ? [XYZ Corp.]
Organization Unit Name ? [Research Dept]
Common Name ? [ TEST.RES.XYZ.COM ]
Email Address ? [webmaster@TEST.RES.XYZ.COM]
Display the Certificate ? [N] Y
```

The certificate request is generated after responding to the last question.

2. View the details of the self-signed certificate (if you chose to display the certificate):

-Version SSL 3.0 protocol
-Serial number Certificates issued by a CA have a serial number that is unique to the certificates issued by that CA.
-Signature Algorithm
-Issuer Your **distinguished name**
-Validity (inception and expiration dates)
-Public key information



Create a certificate authority

Creating a certificate authority (CA) means you can issue certificates using your own private key. The corresponding CA public key is itself contained within a certificate, called a CA Certificate. You must distribute this certificate to clients for them to access your server. A browser must contain this CA Certificate in its "trusted root library" in order to "trust" certificates signed by the CA's private key.

Do the following:

1. Enter the required information to create a certificate authority:

   -PEM Pass Phrase?
   -Confirm PEM Pass Phrase?

   -Encryption Bits? 1024 bits is the largest recommended size.

Explanation: Encryption strength is often described in terms of the size of the keys used to perform the encryption: in general, longer keys provide stronger encryption. Key length is measured in bits. Private key sizes larger than 1024 bits are incompatible with some versions of Netscape Navigator and Microsoft Internet Explorer.

-Default Days? The default number of days until expiration for certificates issued by the CA.

-Certificate Key File? Use OpenVMS syntax (usually, `OPENSSL_ROOT:[KEY]SERVER_CA.KEY`)
-Certificate File? Use OpenVMS syntax (usually, `OPENSSL_ROOT:[CRT]SERVER_CA.CRT`)

-Country Name? The remaining questions determine your server's **Distinguished Name**

Usage note: A Certificate Authority may define a policy specifying which distinguished names are optional and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the common name for a certificate representing a server has a name that matches a wildcard pattern for the domain name of that server, such as *.xyz.com. **Source:** mod_ssl Documentation

-State or Province Name?

-City Name?
-Organization Name?
-Organization Unit Name?

-Common Name? This must be the same as your server's DNS host name (or virtual host name, if name-based virtual hosting is used).

Explanation: Browsers compare the common name in the server certificate with the host name of the server they are connecting to. These must match.

-Email Address?
-Display the Certificate?

Important: All fields must be completed to create a valid certificate request.

```
                    OpenSSL Certificate Tool

                        Create Certificate Authority

PEM Pass Phrase ? []
Confirm PEM Pass Phrase ? []
Encryption Bits ? [1024]
Default Days ? [30] 100
Certificate Key File ? [OPENSSL_ROOT:[KEY]SERVER_CA.KEY]
Certificate File ? [OPENSSL_ROOT:[CRT]SERVER_CA.CRT]
Country Name ? [US]
Organization Name ? [] XYZ Corp.
Organization Unit Name ? [] Research Dept.
Common Name ? [CA Authority] XYZ Corp.
Display the Certificate ? [N] y
```

The certificate request is generated after responding to the last question.

2. View the details of the certificate authority (if you chose to display the certificate):

-Version SSL 3.0 protocol
-Serial number Certificates issued by a CA have a serial number that is unique to the

certificates issued by that CA.
-Signature Algorithm
-Issuer Your **distinguished name**
-Validity (inception and expiration dates)
-Public key information

```
                    OpenSSL Certificate Tool

                        Create Certificate Authority

                  < OPENSSL_ROOT:[CRT]SERVER_CA.CRT; >                  Page 1 of 1

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=US, O=XYZ Corp., OU=Research Dept., CN=XYZ Corp.
        Validity
            Not Before: Jul 18 17:44:48 2000 GMT
            Not After : Oct 26 17:44:48 2000 GMT
        Subject: C=US, O=XYZ Corp., OU=Research Dept., CN=XYZ Corp.
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:bd:d2:d2:c7:80:2e:fe:c6:0f:92:8b:ab:38:a6:
                    a6:48:aa:5a:21:f6:ab:b6:eb:60:03:0d:d0:8d:0d:
                    b9:8d:29:97:62:c3:71:c6:ba:4f:c3:69:02:99:5d:
                    d6:5c:ce:07:eb:d4:fd:39:f4:02:06:a0:2a:ad:56:
                    7d:04:81:37:29:6f:12:0e:86:58:73:aa:6d:11:a3:
                    e9:ed:8a:a8:f1:c4:81:75:2b:7c:07:de:08:66:6c:
                    ee:cf:cc:87:02:6c:d4:06:21:29:a2:3d:8b:88:8b:
                    dd:e7:99:33:2e:16:da:cf:ec:84:f7:7c:29:79:28:
                    c5:d0:cc:59:5b:28:2a:b8:25
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        1e:d6:d7:e1:89:46:ba:1e:37:5c:f5:f6:3b:f6:0f:5f:a8:f7:
        c3:d4:b5:43:3a:af:79:00:09:31:89:c3:2f:36:f3:ba:b8:9b:
        10:76:8a:e9:5d:81:dd:e6:72:fc:8f:8d:39:a8:dd:d7:97:fd:
        73:f0:fc:ec:4a:09:18:14:b3:05:0f:02:27:57:e2:b2:cf:70:
        b1:2e:06:52:d9:ba:0a:18:d3:81:d1:f7:ab:27:66:df:6d:70:
        e2:c7:90:7f:7b:ad:55:dd:0d:36:59:11:13:62:d2:cd:03:82:
        8f:5e:56:ce:00:a3:86:1d:58:38:bf:f4:30:f2:5c:83:82:92:
        e6:d3

                  Enter B for Back, N for Next, Ctrl-Z to Exit
```

Sign a certificate request

Signing someone else's certificate request is the function of a certificate authority. When you send the requested certificate back to them, they start their server using the signed certificate and the pass phrase they have. Embedded in the certificate is your public key. It must match the public key you distribute to clients using this server.

1. Enter the required information to sign a certificate by specifying the following:

-Certificate File specification Use OpenVMS syntax (usually, `OPENSSL_ROOT:[CRT]SERVER.CA.CRT`)
-Certificate Key File specification Use OpenVMS syntax (usually, `OPENSSL_ROOT:[KEY]SERVER_CA.KEY`)
-Certificate Request File? Use OpenVMS syntax (usually, `OPENSSL_ROOT:[CSR]SERVER.CSR`)
-Signed Request File specification Use OpenVMS syntax (usually, `OPENSSL_ROOT:[CRT]SIGNED.CRT`)
-Default Days The default number of days until the signed certificate expires.
-PEM Pass Phrase This is a verification field only. You must use the same pass phrase you used to create the certificate authority (Option 5).

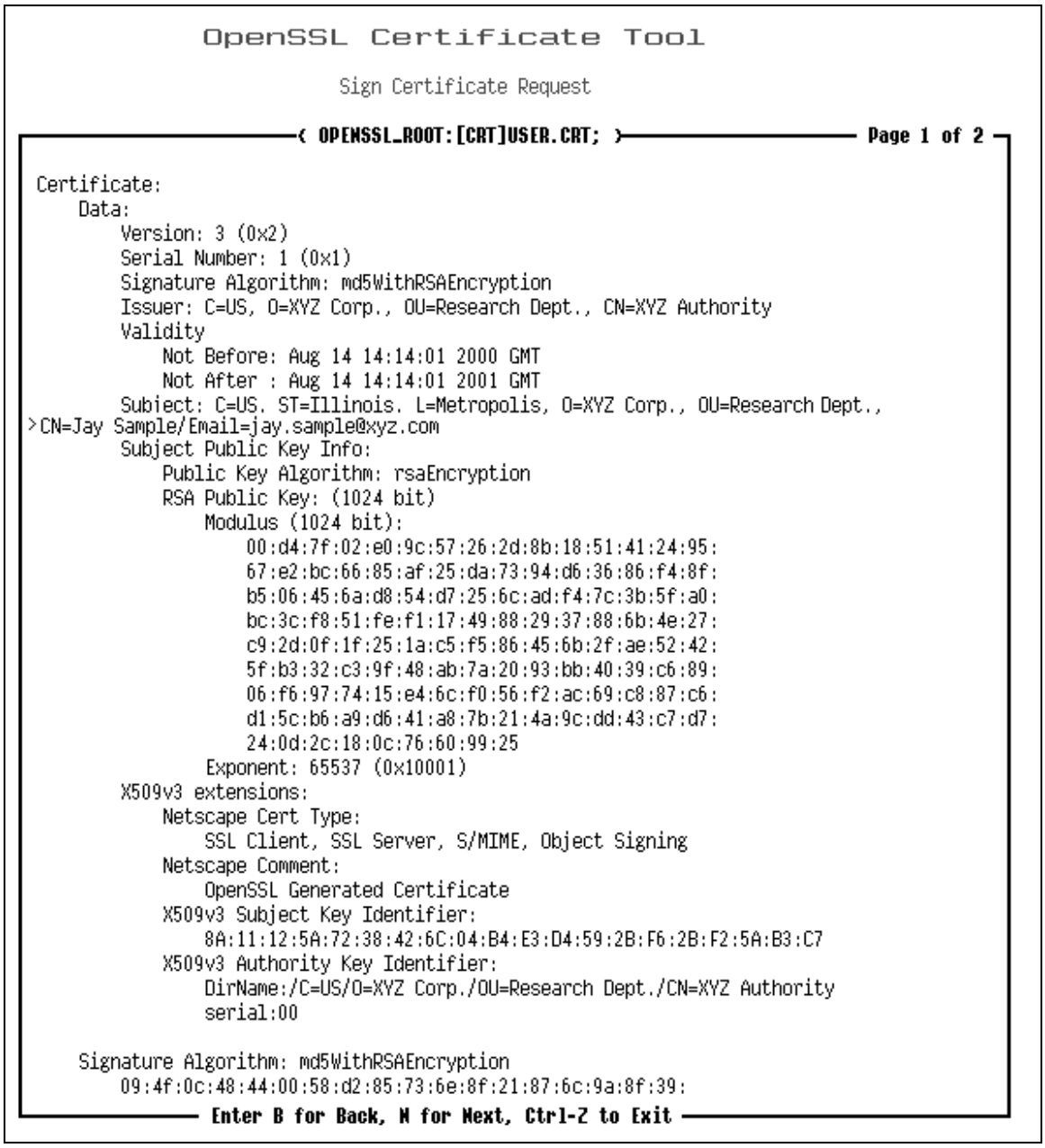
Important: The inception time of a certificate is based on UTC (Coordinated Universal Time). Check with your system administrator that your computer's UTC is set correctly.
Setting Correct Time Zone Information on Your System

The certificate is signed after responding to the last question.

```
                    OpenSSL Certificate Tool

                          Sign Certificate Request

Certificate File ? [OPENSSL_ROOT:[CRT]SERVER_CA.CRT]
Certificate Key File ? [OPENSSL_ROOT:[KEY]SERVER_CA.KEY]
Certificate Request File ? [OPENSSL_ROOT:[CSR]SERVER.CSR] USER
Signed Request File ? [OPENSSL_ROOT:[CRT]SIGNED.CRT] USER
Default Days ? [365]
PEM Pass Phrase ? []
Display the Certificate ? [N] y
```

2. View the details of the signed certificate (if you chose to display the certificate):

-Version SSL 3.0 protocol
-Serial number Certificates issued by a CA have a serial number that is unique to the certificates issued by that CA.
-Signature Algorithm
-Issuer
-Validity (inception and expiration dates)
-Public key information



Hash certificate authorities

This command is required to PEM-encode third-party certificates files and ones you create using Option 5 (which by default are named `SERVER_CA.CRT`). The mod_ssl directives related to CA certificate management (**SSLCACertificatePath** and **SSLCACertificateFile**) require hashed files in order to work.

1. Enter the path in which you have installed your CA files.

   By default, this is: APACHE`$ROOT:[CONF.SSL_CRT]*.CRT`

   Press the Return key to hash the CA files at the specified location.

   This example would hash the *.CRT files found in the system-specific configuration. If you

wanted to hash files for a common configuration, you would use `APACHE$COMMON` instead.

You can verify the existence of the hashed file in the directory you selected. For example,

```
$ DIR APACHE$COMMON:[CONF.SSL_CRT]

Directory APACHE$COMMON:[CONF.SSL_CRT]

AE0FEDEE.0;4 DELETE_HASH_FILES.COM;1 SERVER_CA.CRT;4

Total of 3 files.
```

Hash certificate revocations

This command is required to PEM-encode third-party certificates revocation lists (CRLs) and ones you create using the OpenSSL command line. The mod_ssl directives related to managing client revocation lists (**SSLCARevocationPath** and **SSLCARevocationFile**) require hashed CRL files in order to work.

1. Install a trusted root CA's CRL file or create your own using the `$ OPENSSL CA` command (see *How to use command-line OpenSSL*).
2. Enter the path in which you have installed your CRL files.

   By default, the location is: `APACHE$ROOT:[CONF.SSL_CRL]*.CRL`

   Press the Return key to hash the CRL files at the specified location.

```
                    OpenSSL Certificate Tool

                         Hash Certificate Revocations

Certificate Revocation Path: ? [APACHE$ROOT:[CONF.SSL_CRL]*.CRL]
```

   This example would hash the *.CRL files found in the system-specific configuration. If you wanted to hash files for a common configuration, you would use APACHE$COMMON instead.

   You can verify the existence of the hashed file in the directory you selected. For example,

```
$ DIR APACHE$SPECIFIC:[CONF.SSL_CRL]

Directory APACHE$SPECIFIC:[CONF.SSL_CRL]

AE0FEDEE.R0       CA-BUNDLE.CRL     DELETE_HASH_FILES.COM

Total of 3 files.
```

Using Certificates

This chapter tells you how to put certificates to work on your SSL-enabled *HP Secure Web Server*. There are instructions that will show you how to use **mod_SSL**, **OpenSSL**, and the **Certificate Tool** to set up your server's security. Those commands that require command-line OpenSSL are introduced and explained at the end of this chapter.

How to use certificates

A self-signed certificate is automatically generated for your server when you run *SWS* in SSL mode. In a production environment you will need to pay for a commercial CA to sign your certificate request so that clients will automatically trust your site.

How to test a real server certificate

You can test a real server certificate by using a CA's trial program. For example, you can test Verisign's Secure Server ID.

Follow these steps to install a CA's certificate (also referring to your CA's instructions as they apply to Apache with mod_ssl):

1. In the **OpenSSL Certificate Tool** generate a Certificate Request (using the default responses in most cases).
2. Send the generated file *.CSR file or the contents of the file to the CA by secure email or whatever submission process is provided.

To copy the .CSR file contents, exit the configuration utility, and use VIEW or EDIT to copy the contents.

3. Receive the digitally signed certificate file by secure email or another means.
4. After making backups, replace the existing *.CRT file or replace its contents with the new one. Also replace the existing *.KEY file with the new one that was generated with the certificate request (but not sent to the CA). The SERVER.CSR file is no longer needed.

   To copy the files:

   ```
   $ COPY APACHE$SPECIFIC:[OPENSSL.CRT]SERVER.CRT
   APACHE$SPECIFIC:[CONF.SSL_CRT]
   ```

   ```
   $ COPY APACHE$SPECIFIC:[OPENSSL.KEY]SERVER.KEY
   APACHE$SPECIFIC:[CONF.SSL_KEY]
   ```

   To edit the CRT file (first make it writable):

   ```
   $ EDIT APACHE$ROOT:[CONF.SSL_CRT]SERVER.CRT
   ```

   ! Before pasting the contents of the new certificate, make sure you eliminate line breaks (caused by some mail programs) if necessary by pasting into a text editor first.

5. Restart the server.

   ```
   $ @SYS$STARTUP:APACHE$SHUTDOWN.COM
   ```

   ```
   $ @SYS$STARTUP:APACHE$STARTUP.COM
   ```

6. Test your new server certificate in a client browser using the http**s**:// prefix.
7. You should receive a security alert because the site certificate of the root CA

corresponding with the trial server certificate will not be in your browser.

On installing a paid-for version of a Verisign certificate, such a warning would not be shown because its root CA site certificate would already be in the certificate store of your browser.

Important: You have secured the web server, but this security only applies to those pages that clients access using http**s**://. Pages accessed with the standard http:// are not secure. Therefore to implement a secure site or a site with secure and unsecured pages, you must specify which pages may only be viewed with a secure connection.

How to install a Verisign Global Server ID

Please be aware of the client requirements (below) before installing a Verisign Global Server ID (GSID).

The following instructions configure the Global Server ID server certificate in the system-specific configuration directory and the Intermediate CA certificate in the common configuration directory. If this is not appropriatefor your site, the location of the Intermediate CA certificate can be placed in the system-specific configuration directory.

1. **Obtain a Global Server ID from VeriSign.**

   Enroll at http://www.verisign.com/server/enroll/globalIntro.html.

   Generate a certificate request file.

2. **Download and install the Intermediate CA Certificate.**

   Click the link for Intermediate-CA.

   Cut and paste the entire text of the Intermediate CA certificate, including `the -----BEGIN CERTIFICATE-----` and `-----END CERTIFICATE-----` lines, into a file named:

   ```
   APACHE$COMMON:[CONF.SSL_CRT]INTERMEDIATE-CA.CRT
   ```

   ! Before pasting the contents of the new certificate, make sure you eliminate line breaks (caused by some mail programs) if necessary by pasting into a text editor first.

   Add the following directive to your `APACHE$COMMON:[CONF]HTTTPD.CONF` file, within the `<VirtualHost>` section that defines your secure Web server and with the other SSL directives:

   ```
   SSLCCertificateChainFile
      /apache$common/conf/ssl_crt/intermediate-ca.crt
   ```

3. **Install the Server Certificate**.

   Place the server certificate you received from Verisign and key you generated in the certificate directory:

   ```
   APACHE$SPECIFIC:[CONF.SSL_CRT]SERVER.CRT

   APACHE$SPECIFIC:[CONF.SSL_KEY]SERVER.KEY
   ```

   Add the following directives to your APACHE$COMMON:[CONF]HTTTPD.CONF file, within the `<VirtualHost>` section that defines your secure Web server and with the other SSL directives:

   ```
   SSLCertificateFile /apache$specific/conf/ssl_crt/server.crt

   SSLCertificateKeyFile /apache$specific/conf/ssl_key/server.key
   ```

4. Restart the server.

   ```
   $ @SYS$STARTUP:APACHE$SHUTDOWN.COM
   ```

```
$ @SYS$STARTUP:APACHE$STARTUP.COM
```

You are now using your Global Server ID.

Client requirements
Global Server IDs will work with the following browsers:

Netscape Navigator 4.0 or later

Microsoft Internet Explorer 4.0 or later

Microsoft Internet Explorer 3.02 or later on Windows NT

If your users are using Microsoft Internet Explorer 3.02 on Windows 95, they will need to install a free patch (English Exportable SGC Add-On for IE 3.02).

If your users are using Netscape Navigator 3.0, they will be able to connect to your site at the 40-bit encryption. Navigators prior to 3.0 or Internet Explorers prior to 3.02 will not work with GSIDs.

How to enforce secure pages selectively

It's important to realize that installing a trusted CA certificate does not enforce blanket security for your server unless you require it. You may want to specify which of your server's directories or files require a secure connection. Without doing so, clients are able to view the same pages using URLs beginning with http:// as well as https://.

The simplest way to do this is by using the **SSLRequireSSL** directive in the HTTPD.CONF file. If you apply it to the HTDOCS (or equivalent) directory, it prevents access to any pages in that directory or subdirectories without a secure connection (without using https://).

You can also include the SSLRequireSSL directive in .HTACESS files for individual directories. Using HTTPD.CONF is the more secure method, but this requires stopping and restarting the server. Using an .HTACCESS file offers greater flexibility but also has the potential to compromise performance and security.

In order to enforce authentication of all clients, use the **SSLVerifyClient** directive. The **require** option makes the presentation of a client ID mandatory.

A much more complex directive, **SSLRequire**, enables you to implement selective security using client verification on a per directory basis. You construct SSLRequire directives using Boolean statements that parse the credentials of client certificates (using their corresponding environment variables). The official mod_ssl documentation explains how to construct such directives.

See also

How to use the **FakeBasicAuth** option (below).

How to create and distribute client certificates

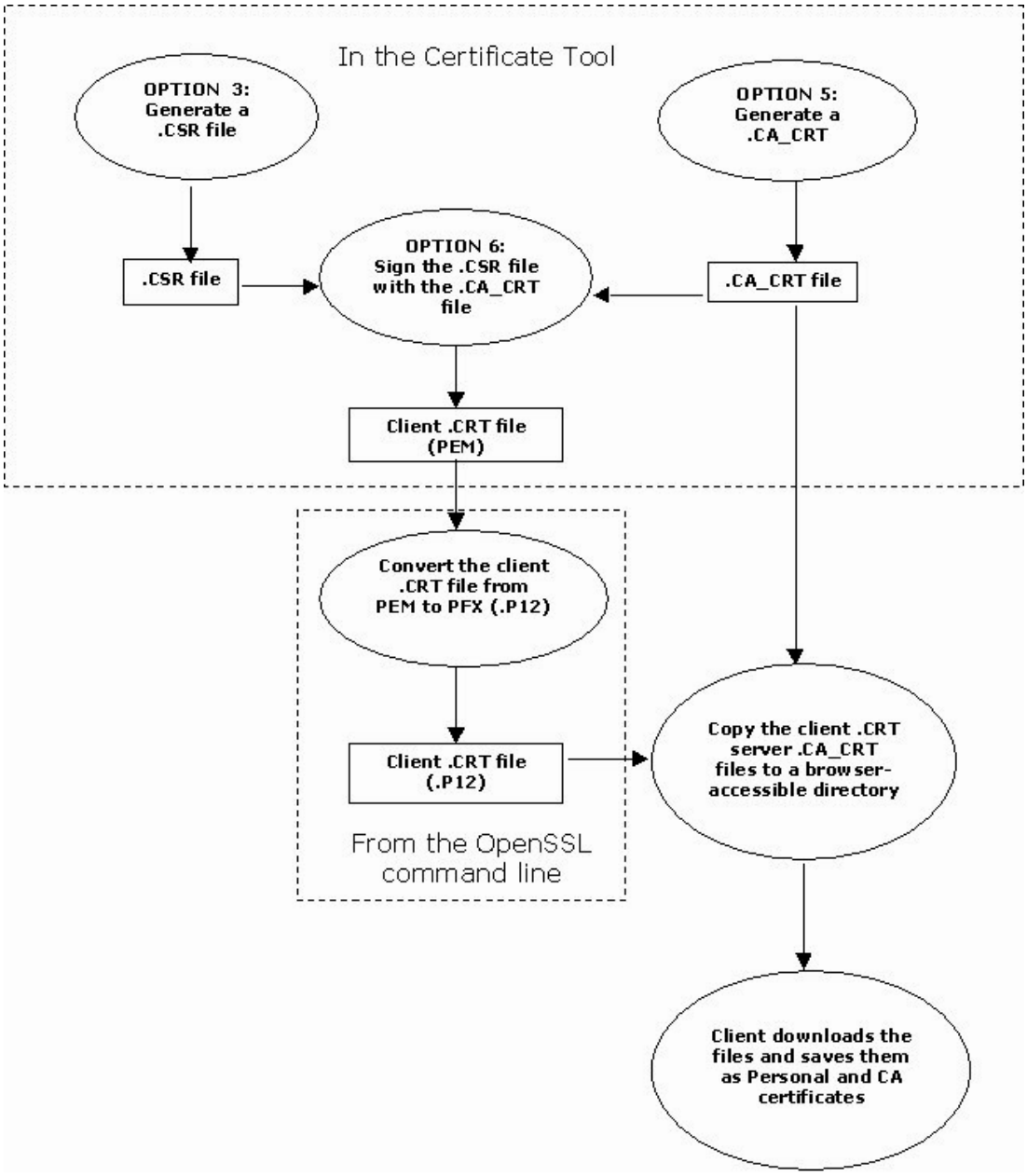
Issuing client certificates means you are performing the role of a Certificate Authority with the purpose of requiring clients to use a certificate that you issue. The following steps are involved:

1. Using the **Certificate Tool**, create a certificate request (Option 3) and sign the certificate with your CA certificate (Option 6).

   Option 6 assumes you have already generated a CA certificate file (Option 5) for your server (default file name is `SERVER_CA.CRT`).

   Client certificates are issued to individual persons. Therefore the **common name** is the individual's proper name (not the name of a network node).

Important: When signing the client certificate you must use the same pass phrase you used to create your certificate authority.



2. Convert the signed client certificate from PEM format to PKCS12 format by using the following command from the OpenSSL command line:

```
$ openssl pkcs12 -export -in <CLIENT_NAME>.CRT
-inkey OPENSSL_KEY:<CLIENT_NAME>.KEY
-out <CLIENT_NAME>.P12 -name "<Issuer Name>"
```

   The -out parameter should be a full file specification if you have not SET DEFAULT to the same directory as the .CRT file.

   For example:

```
$ openssl pkcs12 -export -in JSample.CRT
-inkey OPENSSL_KEY:JSample.KEY
-out JSample.P12 -name "XYZCorp"

Enter Export Password:
```

```
Verifying password - Enter Export Password:
```

The export password that you specify is required by the recipient of the certificate when installing it.

3. Distribute the certificate from the server to the client's browser.

   A client can receive a certificate by email or directly using the browser. In the case of Internet Explorer (IE), having clients point directly to the files is the simplest way. This method may also be used with Netscape Navigator, but not with the advantage of automating the process.

   If you use the browser method, copy the client certificate and the server certificate to your HTDOCS (or another accessible) directory. Clients can then point their browsers at the certificate files and save them. For example, the URLs could be:

   ```
   http://test.res.xyz.corp/martian_client.p12
   ```

   and

   ```
   http://test.res.xyz.corp/server_ca.der
   ```

Important: In order to serve PKCS12 client certificates correctly to a Netscape users, you need to define this file type in HTTPD.CONF (see below).

In the case of IE, opening the save file will start the **Certificate Manager Import Wizard** automatically.

In the case of Netscape Navigator, users should load them using the **Security Info** window.

A. From the **Communicator** menu, choose **Tools**, and then choose **Security Info**.

B. From the **Security Info** window, click **Certificates**: **Yours** and then click **Import a Certificate**.

Important: Clients must load both the client certificate and the server certificates. The password you use when converting the certificates to PKCS12 format is required by clients to install the certificates.

4. On the server, edit your HTTPD.CONF file to uncomment **SSLVerifyClient**, giving it the value of **Require**. Also uncomment **SSLVerifyDepth**, leaving the value of 10 under most circumstances.

5. Stop and restart the server.

   ```
   $ @SYS$STARTUP:APACHE$SHUTDOWN
   $ @SYS$STARTUP:APACHE$STARTUP
   ```

How to add PCS12 file type to MOD_SSL.CONF

Unless you define the PCS12 file type on your server, Netscape browsers will not be able to save certificate files with a .P12 extension. To specify the file type add the following to the MOD_SSL.CONF file under MIME-types, either inside or outside the <IfDefine SSL> section, and restart the server:

```
AddType application/octet-stream .p12
```

This will cause Netscape browsers to display the **Save As** dialog for this file type.

How to implement the FakeBasicAuth option

This is an option of the **SSLOptions** directive. Using this option causes *HP Secure Web Server*

to use standard Apache authentication based on the client certificate's **distinguished name**.

1. Create a password file containing the following line for each client certificate:

   ```
   <Distinguished Name fields of a certificate>:xxj31ZMTZzkVA
   ```

   where:

   - <Distinguished Name fields of a certificate> is required for every client certificate. You can obtain these by using the following OpenSSL command line:

     ```
     $ openssl x509 -noout -subject -in "client certificate"
     ```

   - "xxj31ZMTZzkVA" is the literal string you should use.

     This is predefined DES-encrypted string (actually, the word "password") for any client certificate used with **FakeBasicAuth**.

2. Define a user authentication scheme in HTTPD.CONF or access files (.HTACCESS).

   For example, the definition could be as follows in HTTPD.CONF:

   ```
   <Directory /apache$common/htdocs>

   SSLRequireSSL
   SSLVerifyClient require
   SSLOptions +FakeBasicAuth +StrictRequire
   AuthName "FakeBasicAuth Client Authentication"
   AuthType Basic
   AuthUserFile /apache$common/conf/ssl/fba_passwd.txt
   require valid-user
   </Directory>
   ```

   In an access file, omit the <Directory> section command (first and last line).

3. Restart the server if you added the definition to HTTPD.CONF.

FAQs

Why do I already have a server certificate on my system?

Self-signing a certificate is a required step before starting *HP Secure Web Server* if you've enabled SSL. This step is performed for you when you run the SWS configuration tool:

```
$ @SYS$MANAGER:APACHE$CONFIG.COM
```

You can examine the file's contents by choosing Option 1 in the **OpenSSL Certificate Tool** and entering the default specification:

```
APACHE$ROOT:[CONF.SSL_CRT]SERVER.CRT
```

Your SSL-aware server will not run without a valid certificate. However, a certificate does not have to be signed by a public CA. Self-signing means that you have used your private key to sign the certificate, which in turn contains your public key. Clients now have the option of choosing to install your self-signed certificate as a trusted root CA certificate.

Can I install more than one server certificate?

Yes. Multiple server certificates for virtual hosts need to be defined using individual **SSLCertificateFile** and **SSLCertificateKeyFile** directives.

How to use command-line OpenSSL

SSL-enabled HP Secure Web Server includes the complete OpenSSL command-line interface in its native UNIX format. Whether you will need to use this depends on the type of administrative tasks you plan to do. For example, if you are implementing client authentication, one requisite activity is to generate a Client Revocation List if you are issuing client certificates.

Start the OpenSSL command-line interface with this command:

```
$ @APACHE$COMMON:[OPENSSL.COM]OPENSSL_INIT_ENV.COM
```

Then enter the following, to choose a directive and proceed:

```
$ OPENSSL <commandname>
```

If you type an unknown command name, a complete list of commands (standard, message digest, and cipher) is displayed.

```
$ OPENSSL <unknown_commandname>
```

```
$ openssl ?
openssl:Error: '?' is an invalid command.

Standard commands
asn1parse      ca             ciphers        crl            crl2pkcs7
dgst           dh             dhparam        dsa            dsaparam
enc            errstr         gendh          gendsa         genrsa
nseq           passwd         pkcs12         pkcs7          pkcs8
rand           req            rsa            s_client       s_server
s_time         sess_id        smime          speed          spkac
verify         version        x509

Message Digest commands (see the `dgst' command for more details)
md2            md5            mdc2           rmd160         sha
sha1

Cipher commands (see the `enc' command for more details)
base64         bf             bf-cbc         bf-cfb         bf-ecb
bf-ofb         cast           cast-cbc       cast5-cbc      cast5-cfb
cast5-ecb      cast5-ofb      des            des-cbc        des-cfb
des-ecb        des-ede        des-ede-cbc    des-ede-cfb    des-ede-ofb
des-ede3       des-ede3-cbc   des-ede3-cfb   des-ede3-ofb   des-ofb
des3           desx           rc2            rc2-40-cbc     rc2-64-cbc
rc2-cbc        rc2-cfb        rc2-ecb        rc2-ofb        rc4
rc4-40         rc5            rc5-cbc        rc5-cfb        rc5-ecb
rc5-ofb
```

How to create and view a client revocation list

If you want to implement a client revocation list using the mod_SSL directives, **SSLCARevocationPath** and **SSLCARevocationFile**, you will need to do set up your list using OpenSSL commands in the following way:

Create the client revocation list

The format of this command is as follows:

```
$ openssl ca -gencrl -config OPENSSL_CA.CONF -revoke
<FILESPEC>.CRT -out <FILESPEC>.CRL
```

Notes:

- OpenSSL arguments (shown lowercase) may precede or proceed OpenVMS file specifications (shown uppercase).

- If you do not have default_crl_days defined in your OPENSSL_CA.CONF file, you must supply this on the command line also (as in the following example).
- If you don't specify otherwise, the command expects to find the client certificate in APACHE$COMMON:[CONF].
- If you get an error message "Unable to load 'random state'," you can create a RANDFILE environment variable, as follows:

```
$ SHOW SYSTEM /FULL /OUT=SYS$LOGIN:RANDFILE.;

$ DEFINE /PROCESS RANDFILE SYS$LOGIN:RANDFILE.;
```

Example:

```
$ openssl ca -gencrl -config OPENSSL_CA.CONF -revoke
JAY_SAMPLE.CRT -out CA-BUNDLE.CRL -crldays 365

Using configuration from openssl_ca.conf

Enter PEM pass phrase: <phrase>

Revoking Certificate 03.

Data Base Updated
```

The files specified are the CA configuration file (OPENSSL_CA.CONF), the client certificate file (JAY_SAMPLE.CRT), and the CRL file (CA-BUNDLE.CRL).

View the client revocation list

The format of this command is as follows:

```
$ openssl crl -in <FILESPEC>.CRL -text -noout
```

Example:

This command would open the CRL file created by the previous example.

```
$ openssl crl -in APACHE$ROOT:[CONF.SSL_CRL]CA-BUNDLE.CRL -text -
noout

Certificate Revocation List (CRL):

  Version 1 (0x0)
  Signature Algorithm: md5WithRSAEncryption
  Issuer: /C=US/O=XYZ Corp./OU=Research Dept./CN=XYZ Authority
  Last Update: Aug 14 16:27:42 2004 GMT
  Next Update: Aug 14 16:27:42 2005 GMT
No Revoked Certificates.
  Signature Algorithm: md5WithRSAEncryption
  83:47:e1:ce:f9:d9:41:ef:29:e7:a8:90:66:ee:1b:ad:50:37:
  bf:d3:16:ec:14:52:e5:1c:4f:dc:95:46:5b:ba:28:73:87:8f:
  3f:49:80:11:08:8b:ab:64:56:77:bf:9f:75:3a:d7:be:55:a9:
  87:2f:58:c2:59:80:31:52:a4:7d:28:00:24:a6:cc:0d:23:a2:
  00:5c:f5:04:f5:91:80:59:ab:52:dc:72:83:ac:40:40:1b:08:
  fa:bd:d0:f9:c4:45:47:7a:c0:52:0b:3a:22:e4:5e:2a:8d:5d:
  fa:74:f1:1b:ee:ec:ce:88:c5:c6:50:4a:e2:74:9b:96:9f:cb:
  f6:a8
```

FAQs

After entering OPENSSL -?, why am I prompted for a _File?

You should use the following command to work with the OpenSSL command line:

```
$ @APACHE$COMMON:[OPENSSL.COM]OPENSSL_INIT_ENV.COM
```

After doing this, you can proceed by entering `$ OPENSSL` once or prior to each command.

Where are the OPENSSL configuration files?

OpenSSL configuration files can exist in the system-specific or common CONF directory.

When using common configuration files across a cluster:

`APACHE$COMMON:[CONF]OPENSSL.CONF and OPENSSL_CA.CONF`

When using system-specific configuration files:

`APACHE$SPECIFIC:[CONF]OPENSSL.CONF` and `OPENSSL_CA.CONF`

How do I view certificates and certificate requests?

If you don't want to use the **Certificate Tool** for this purpose, use the following commands from the OpenSSL command line:

To view a certificate request:

```
$ OPENSSL REQ -IN <FILE_NAME>.CSR -NOOUT -TEXT
```

For example:

```
$ OPENSSL REQ -IN [.OPENSSL.CSR]MR.CSR -NOOUT -TEXT
```

```
To view a certificate:
```

```
$ OPENSSL X509 -IN <INPUT_FILE>.CRT -NOOUT -TEXT
```

For example:

```
$ OPENSSL X509 -IN [.OPENSSL.CRT]MR.CRT -NOOUT -TEXT
```

Why and how do I convert from PEM to DER and PFX formats?

These formats are methods of hashing certificates for distribution to clients. From the OpenSSL command line, use the following commands:

To convert to **DER**:

```
$ openssl X509 -in <FILE_NAME>.PEM -inform PEM -outform DER -out
<FILE_NAME>.DER
```

To convert to **PFX** (Personal Information Exchange or **PKCS12**) format:

```
$ openssl PKCS12 -export -in KEVIN_CLIENT.CRT
```

```
-INKEY OPENSSL_KEY:KEVIN_CLIENT.KEY
```

```
-OUT KEVIN_CLIENT.P12 -NAME "Your Name"
```

Glossary of SSL-related terms

Certificate (*aka* Digital Certificate)

A data record used for authenticating network entities such as a server or a client. A certificate contains X.509 information pieces about its owner (called the subject) and the signing *Certificate Authority* (called the issuer), plus the owner's public key and the signature made by the CA. Network entities verify these signatures using CA certificates.

Certificate (*aka* Certification) Authority (CA)

A trusted third party whose purpose is to sign certificates for network entities it has authenticated using secure means. Other network entities can check the signature to verify that a CA has authenticated the bearer of a certificate.

Certificate Signing Request (CSR)

An unsigned certificate for submission to a *Certification Authority*, which signs it with the *Private Key* of their CA *Certificate*. Once the CSR is signed, it becomes a real certificate.

Cipher

An algorithm or system for data encryption. Examples are DES, IDEA, RC4, etc.

Configuration Directive

Most Apache configuration directives are in the HTTPD.CONF file.

Digital Signature

An encrypted text block that validates a certificate or other file. A *Certification Authority* creates a signature by generating a hash of the *Public Key* embedded in a *Certificate*, then encrypting the hash with its own *Private Key*. Only the CA's public key can decrypt the signature, verifying that the CA has authenticated the network entity that owns the *Certificate*. ***See also****,* Hash Function *and* Message Digest.

Distinguished Name

A DN is a series of name-value pairs, such as uid=doe, that uniquely identifies the certificate **subject.**

Fully-Qualified Domain-Name (FQDN)

A hostname and a domain name that can resolve to an IP address (for example, www.hp.com).

Hash Function

A fixed-length value created mathematically to identify data uniquely.

Message Digest

A hash of a message, which can be used to verify that the contents of the message have not been altered in transit. This principal is employed in digital signatures.

OpenSSL

The Open Source toolkit for SSL/TLS; see http://www.openssl.org/

Pass Phrase

The word or phrase that protects private key files. It prevents unauthorized users from encrypting them. Usually it's just the secret encryption/decryption key used for *Ciphers*.

PEM (Privacy Enhanced Mail)

A standard, predating S/MIME, for encrypting e-mail and authenticating senders.

Private Key

The secret key in a *Public Key Cryptography* system, used to decrypt incoming messages and sign outgoing ones.

Public Key

The publically available key in a *Public Key Cryptography* system, used to encrypt messages bound for its owner and to decrypt signatures made by its owner.

Public Key Cryptography

The study and application of asymmetric encryption systems, which use one key for encryption and another for decryption. A corresponding pair of such keys constitutes a key pair. Also called Asymmetric Crypography.

Secure Sockets Layer (SSL)

A protocol created by Netscape Communications Corporation for general communication authentication and encryption over TCP/IP networks. The most popular usage is *HTTPS* - HyperText Transfer Protocol (HTTP) using SSL.

SSLeay

The original SSL/TLS implementation library developed by Eric A. Young http://www.ssleay.org/

Symmetric Cryptography

The study and application of *Ciphers* that use a single secret key for both encryption and decryption operations.

Transport Layer Security (TLS)

The successor protocol to SSL, created by the Internet Engineering Task Force (IETF) for general communication authentication and encryption over TCP/IP networks. TLS version 1 and is nearly identical with SSL version 3.

X.509

The most widely used standard for digital certificates. It is recommended by the International Telecommunication Union (ITU-T) and is used for SSL/TLS authentication.