# Building a Secure RedHat Apache Server HOWTO

# Table of Contents

**Building a Secure RedHat Apache Server HOWTO**......1
- Richard Sigle, Richard.sigle@equifax.com......1
- 1. Purpose/Scope of this Guide......1
- 2. Introduction to Secure Sockets Layer/Private Key Infrastructure......1
- 3. Working with Certificates......1
- 4. Configuring your Apache Server......1
- 5. Troubleshooting......1
- 6. Glossary......2
- 1. Purpose/Scope of this Guide......2
- 1.1 About Secure Sockets Layer (SSL)......2
- 1.2 FeedBack......2
- 1.3 Copyrights and Trademarks......3
- 1.4 Acknowledgements and Thanks......3
- 2. Introduction to Secure Sockets Layer/Private Key Infrastructure......3
- 2.1 Responsibilities of SSL/PKI......3
- 2.2 How SSL Works......4
  - SSL Handshake Protocol......4
  - Session Key(Symmetric Code)......5
  - Public/Private Key Pair(Asymmetric Code)......5
- 2.3 How PKI Works......5
- 2.4 Certificates(x509 Standard)......6
- 2.5 Digital Certificate Private Key......6
- 2.6 Digital Certificate Public Key......7
- 2.7 Certificate Signing Request(CSR)......7
- 3. Working with Certificates......7
- 3.1 Create a Private Key......7
- 3.2 Create a Certificate Signing Request......8
- 3.3 Creating a Self−Signed Certificate......8
- 3.4 Installing your Web Server Certificate......9
- 4. Configuring your Apache Server......9
- 4.1 Define a Secure Virtual Host......9
  - SSL Engine......10
  - SSLCertificateFile......10
  - SSLCertificateKeyFile......10
  - SSLCACertificateFile......11
- 4.2 Certificate Examples......11
  - Server Certificate File......11
  - Contents of the Certificate File......11
  - Private Key File......12
  - Contents of the Private Key......12
- 4.3 Restart the Web Server......13
- 5. Troubleshooting......14
- 5.1 Server Appears to start, but you cannot access the secure site......14
- 5.2 Certificate Name Check Warning is issued by the client's browser......15
- 5.3 Certificate was Signed by an Untrusted Certificate Authority Warning is issued by the client's browser......15
- 5.4 SSLEngine on is an un−recognized command (when starting Apache)......15
- 5.5 You have forgotten your "PEM Passphrase" and you would like to know how to reset it......15

# Table of Contents

6. Glossary.....15

# Building a Secure RedHat Apache Server HOWTO

## Richard Sigle, `Richard.sigle@equifax.com`

0.1, 2001−02−06

---

*The guide is designed to explain how PKI and SSL work together. It is essential to understand how the SSL protocol works to successfully deploy a secure server.*

---

## 1. Purpose/Scope of this Guide

- 1.1 About Secure Sockets Layer (SSL)
- 1.2 FeedBack
- 1.3 Copyrights and Trademarks
- 1.4 Acknowledgements and Thanks

## 2. Introduction to Secure Sockets Layer/Private Key Infrastructure

- 2.1 Responsibilities of SSL/PKI
- 2.2 How SSL Works
- 2.3 How PKI Works
- 2.4 Certificates(x509 Standard)
- 2.5 Digital Certificate Private Key
- 2.6 Digital Certificate Public Key
- 2.7 Certificate Signing Request(CSR)

## 3. Working with Certificates

- 3.1 Create a Private Key
- 3.2 Create a Certificate Signing Request
- 3.3 Creating a Self−Signed Certificate
- 3.4 Installing your Web Server Certificate

## 4. Configuring your Apache Server

- 4.1 Define a Secure Virtual Host
- 4.2 Certificate Examples
- 4.3 Restart the Web Server

## 5. Troubleshooting

- 5.1 Server Appears to start, but you cannot access the secure site
- 5.2 Certificate Name Check Warning is issued by the client's browser

- 5.3 Certificate was Signed by an Untrusted Certificate Authority Warning
- 5.4 SSLEngine on is an un−recognized command (when starting Apache)
- 5.5 You have forgotten your "PEM Passphrase" and you would like to know

## 6. Glossary

---

## 1. Purpose/Scope of this Guide

The purpose of this guide is to assist RedHat Linux users with the installation of server (SSL) certificates using the Apache web server. The goal is to provide a clear procedure that will save time and, in many cases, money!

First, I will cover what you need to know about the SSL protocol and digital certificates. In my experience, building an Apache web server with ModSSL and OpenSSL is the most beneficial software combination. OpenSSL is a general−purpose cryptography library that supports the SSL v2/v3 and TLS v1 protocols. ModSSL is an Apache API module designed to act as an interface between Apache and OpenSSL. The biggest advantage is that all three packages are free.

Then, beginning with Section 4, I will go through the step−by−step procedures for generating keys and installing certificates on a RedHat−Apache server compiled with ModSSL and OpenSSL. The procedures in Section 4 will also work with commercial SSL−server packages such as Stronghold and Raven that are closely related to Apache.

Disclaimer: I am a technical support engineer for Equifax Secure Inc., a Certificate Authority. Therefore, I use Equifax Secure certificates and examples geared towards installing Equifax Secure certificates. However, the instructions will also work with certificates issued by other Certificate Authorities. Since this document was written at my own initiative, Equifax Secure Inc. is neither liable nor accountable for any consequences resulting from the use of these procedures.

*My comments to the reader is in this style (emphasized).*

`Example lines are in plain roman style`.

*Note that extra comments and advice is found in comments within the SGML source.*

### 1.1 About Secure Sockets Layer (SSL)

SSL is a presentation layer service, located between the TCP and the application layer. It is platform and application independent. SSL is responsible for the management of a secure communications channel between the client and server. SSL provides a strong mechanism for encrypting data transferred between a client and a server.

### 1.2 FeedBack

Comments on this guide may be directed to the author (`richard.sigle@equifax.com`).

## 1.3 Copyrights and Trademarks

Please freely copy and distribute this document in any format. It's requested that corrections and/or comments be forwarded to the document maintainer. You may create a derivative work and distribute it provided that you:

- Send your derivative work (in the most suitable format such as sgml) to the LDP (Linux Documentation Project) or the like for posting on the Internet. If not the LDP, then let the LDP know where it is available.
- License the derivative work with this same license or use GPL. Include a copyright notice and at least a pointer to the license used.
- Give due credit to previous authors and major contributors.

If you're considering making a derived work other than a translation, it's requested that you discuss your plans with the current maintainer.

## 1.4 Acknowledgements and Thanks

I would like to thank Tony Villasenor for tirelessly reading my drafts and offering his input and advice. Without Tony, this document would never have been finished.

---

# 2. Introduction to Secure Sockets Layer/Private Key Infrastructure

PKI is an *asymmetric key system* which consists of a public key (which is sent to clients) and a private key (stays local on the server). PKI differs from a *symmetric key system* in which both the client and server use the same key for encryption/decryption.

## 2.1 Responsibilities of SSL/PKI

SSL sets out to fulfill requirements that make it acceptable for use in the transmission of even the most sensitive of transactions, such as credit card information, medical records, legal documents, and e−commerce applications. Each application can choose to utilize some or all of the following criteria depending on the sensitivity and value of the transactions it will be processing.

***Privacy***

> Let's say that a message is to be coded for transmission from A to B. A uses B's public key to encrypt the message. In this way B will be the only person who can decode and read this message using his private key. We cannot however be sure that A is the person who he claims to be.

***Authenticity***

> In order to be sure that A is the person who he claims to be, we want guaranteed authenticity. This requires a slightly more complex coding process. In this case, A's message to B is first encrypted

with A's private key and then with B's public key. B now has to decrypt it first with his private key and then with A's public key. Now B can be sure that A is who he claims to be as nobody else could create a message encrypted with his private key. SSL achieves this with the use of certificates (PKI). A certificate is issued by a *neutral* third party − such as a certificate authority (CA) − and includes a digital signature and/or a time stamp in addition to the public key of the certified party. A self−signed digital certificate can be created by anyone with the correct SSL tools, but self−signed certificates lack the weight of validation performed by a mutually respected neutral third party.

***integrity***

In SSL, integrity is guaranteed by using a MAC (Message Authentication Code) with the necessary hash table functions. Upon generation of a message, the MAC is obtained by applying a hash function and the result is then added to the message. After the message has been received, validity is then checked by comparing the message's embedded MAC with a new MAC computed from the received message. This would immediately reveal messages that have been altered by a third party.

***Non−Repudiation***

Non−repudiation protects both parties from each other during online transactions. It prevents one or the other from saying that they did not send a particular piece of information. Non−repudiation does not allow either party to alter the transaction after it has been made. Digital non−repudiation is the equivalent of signing a contract, in the traditional sense.

## 2.2 How SSL Works

The SSL protocol includes two sub−protocols: the SSL record protocol and the SSL handshake protocol. The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL−enabled server and an SSL−enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client. The server certificate is signed by a Certificate Authority to insure that it is not corrupted and establishes a *chain of trust*.
- 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.

### SSL Handshake Protocol

The Handshake Protocol is used to co−ordinate the state of the client and the server. During the handshake, the following events take place:

- Certificates are exchanged between the client and server (asymmetric keys). The server sends its public key to the client. If the server is set to verify client authentication via a certificate, the client sends its public key to the server. The validity dates on the certificates are verified and they are checked for the digital signature of a trusted certificate authority. If the validity date and/or digital signature are not correct, the browser will issue a warning to the user. The user is then given the option to trust the certificate holder.

- The client then generates a random key (symmetric key). These will be used for encryption and for calculating MACs. They are encrypted using the server's public key and sent to the server. Only the server has the ability to decrypt the new random key. The new symmetric key is used for encrypting the data that is sent between client and server.

  Note: The use of a symmetric key after server−browser authentication greatly enhances subsequent throughput performance.

- A message encryption algorithm and a hash function for integrity are negotiated. This negotiation process could be carried out such that the client presents a list of supported algorithms to the server, which, in turn, selects the strongest cipher available to both of them. Identifiers for the chosen encryption algorithm and hash function are stored in the cipher spec field of the current state for use by the record protocol.
- All of the following fields are set during handshaking: Protocol Version, Session ID, Cipher Suite, Compression Method and two random values ClientHello.random and ServerHello.random.

**Note:** An IP address is required for each SSL connection. Name based virtual hosts are resolved during the application layer. Remember Secure Sockets Layer resides below the application layer.

### Session Key(Symmetric Code)

- 40−bit, originally used only for export purposes
- 56−bit, used by DES
- 64−bit key − used by CAST, 256 times stronger than 56−bit
- 80−bit key − used by CAST, 16 million times stronger than 56−bit (infeasible to break with current technology)
- 128−bit key − used by CAST or RC2, exhaustive key search impossible now and for the foreseeable future

### Public/Private Key Pair(Asymmetric Code)

- 512−bit
- 768−bit
- 1024−bit
- 2048−bit

## 2.3 How PKI Works

The client and the server each have a public key and a private key (the client's browser randomly creates a key pair for the SSL session, unless, a client certificate is held by the client and requested by the server).

The sender uses their private key to encrypt a message. This act authenticates the source of the message. The resulting cipher is encrypted once more with the receiving party's public key. This action provides confidentiality because only the receiving party is able to do the initial decryption of the message using their private key. The receiver uses the sender's public key to further decrypt the encrypted message. Because only the sender has access to their private key, the receiver is assured that the encrypted message originated from the sender.

A message digest is used to verify that neither party or a third party has tampered with or changed the message in any way. A message digest is obtained by applying a hash function (part of the private key known as the fingerprint) to the message. The digest (which is now known as the signature) is attached or appended to the message. The signature's length is constant (no matter how large the file is) and depends on what type of message digest the private key contains (md5 − 128 bit, sha1 − 160 bit, etc). Changing even one bit in the message will change the length of the signature and thus prove that the message has been tampered with.

## 2.4 Certificates(x509 Standard)

Digital certificates make it possible to trust an entity on the Internet. A digital certificate contains the user's credentials, which have been verified by a neutral third−party certificate authority.

A mathematical algorithm and a value (key) are used to encrypt data into an unreadable form. A second *key* is used to decrypt the data, using a complementary algorithm and a related value. The two keys must contain a related value and are known as a *key pair*.

Note: ITU−T Recommendation X.509 [CCI88c] specifies the authentication service for X.500 directories, as well as the X.509 certificate syntax. The certificate is signed by the issuer to authenticate the binding between the subject (user's) name and the user's public key. SSLv3 was adopted in 1994. The major difference between versions 2 and 3 is the addition of the extensions field. This field grants more flexibility as it can convey additional information beyond just the key and name binding. Standard extensions include subject and issuer attributes, certification policy information, and key usage restrictions.

An X.509 certificate consists of the following fields:

- Version
- serial number
- signature algorithm ID
- issuer name
- validity period
- subject (user) name
- subject public key information
- issuer unique identifier (version 2 and 3 only)
- subject unique identifier (version 2 and 3 only)
- extensions (version 3 only)
- signature on the above fields

## 2.5 Digital Certificate Private Key

The private key is not embedded within a digital certificate. The private key does not include any server information. It contains encryption information and a fingerprint. It is generated locally on your system and should remain in a secure environment. If the private key is compromised, a perpetrator essentially has the code to your security system. The transmissions between client and server can be intercepted and decrypted. This type of vulnerability is why it is recommended to create a private key that is encrypted using triple DES technology. The file is then encrypted and password protected making it all but impossible to use without the correct pass phrase.

The security of a transaction is dependent on its private key. Should this key fall into the wrong hands then anyone can easily duplicate it and use it to compromise security. A compromised key could lead to messages meant for the server to be intercepted and manipulated by unscrupulous hackers. A fully secure system must be able to detect impostors and prevent the duplication of keys.

## 2.6 Digital Certificate Public Key

The public key is embedded in a digital certificate, which is sent by the server to a client when a secure connection is requested. This process identifies the server using the certificate. The public key validates the integrity, authenticity, and is also used to encrypt data to create a private data transmission.

## 2.7 Certificate Signing Request(CSR)

A CSR contains the information required by a certificate authority to create the certificate. The CSR contains an encrypted version of the private key's complimentary algorithm, common value, and information that identifies the server. This information includes, but is not limited to, country, state, organization, common name (domain name), and contact information.

---

# 3. Working with Certificates

The following section covers the steps involved in creating the private key file, certificate signing request, and a self−signed certificate. If you plan to obtain a certificate signed by a certificate authority, you will need to create a *certificate signing request (CSR)*. Otherwise, you can create a self−signed certificate.

## 3.1 Create a Private Key

To create a private key, you must have the OpenSSL toolkit installed and configured with Apache. The following examples use the OpenSSL command line tool which is located in the /usr/local/ssl/bin directory by default. The examples assume that the directory containing the OpenSSL command line tool has been added to the $PATH.

To create a private key using the triple des encryption standard (recommended), use the following command:

```
openssl genrsa −des3 −out filename.key 1024
```

You will be prompted to enter and re−enter a pass phrase. If you choose to use triple des encryption, you will be prompted for the password each time you start the SSL server from a cold start. (When using the restart command, you will not be prompted for the password). Some of you may find this password prompt to be a nuisance, especially if you need to boot the system during off−hours. Or, you may believe that your system is already sufficiently secure. So, if you choose not to have a password prompt (hence no triple des encryption), use the command below. If you would rather create just a 512−bit key, then omit the 1024 at the end of the command and OpenSSL will default to 512 bits. Using the smaller key is slightly faster, but it is also less secure.

To create a private key without triple des encryption, use the following command:

```
openssl genrsa −out filename.key 1024
```

To add a password to an existing private key, use the following command:

```
openssl −in filename.key −des3 −out newfilename.key
```

To remove a password from an existing private key, use the following command:

```
openssl −in filename.key −out newfilename.key
```

**Note:** Your private key will be created in the current directory unless otherwise specified. There are 3 easy ways to deal with this. If OpenSSL is in your path, you can run it from the directory that you have designated to store your key files in (default is `/etc/httpd/conf/ssl.key` if you installed Apache using the RPM or `/usr/local/apache/conf/ssl.key` if you installed Apache using the source files). Another solution is to copy the files from the directory where they were created to the correct directory. And, last but not least, you can specify the path when running the command (eg. `openssl genrsa −out /etc/httpd/conf/ssl.key/filename.key 1024`). Doesn't matter how you do it as long as it gets done before you proceed.

For more information on the OpenSSL toolkit check out: OpenSSL Website.

## 3.2 Create a Certificate Signing Request

To obtain a certificate signed by a certificate authority, you will need to create a Certificate Signing Request (CSR). The purpose is to send the certificate authority enough information to create the certificate without sending the entire private key or compromising any sensitive information. The CSR also contains the information that will be included in the certificate, such as, domain name, locality information, etc.

- Locate the private key that you would like to creat a CSR from. Enter the following command:
  ```
  openssl req −new −key filename.key −out filename.csr
  ```
- You will be prompted for Locality information, common name (domain name), organizational information, etc. Check with the CA that you are applying to for information on required fields and invalid entries.
- Send the CSR to the CA per their instructions.
- Wait for your new certificate and/or create a self−signed certificate. A self−signed certificate can be used until you receive your certificate from the certificate authority.

**Note:** Use the following command to create a private key and request at the same time.

```
openssl genrsa −des3 −out filename.key 1024
```

## 3.3 Creating a Self−Signed Certificate

It is not necessary to create a self−signed certificate if you are obtaining a CA−signed certificate. However, creating a self−signed certificate is very simple. All you need is a private key and the name of the server (fully qualified domain name) that you want to secure. You will be prompted for information such as locality information, common name (domain name), organizational information, etc. OpenSSL gives you a great deal of freedom here. The only required field for the certificate to function correctly is the common name (domain name) field. If this is not present or incorrect, you will receive a *Certificate Name Check* warning from your browser.

To create a self−signed certificate:

```
openssl req −new −key filename.key −x509 −out filename.crt
```

## 3.4 Installing your Web Server Certificate

If you followed these instructions so far you shouldn't have any problems at this point. If you sent your CSR to a certificate authority and you have not gotten your certificate back yet, you can take a break now! If you are using a self−signed certificate, or you have received your certificate, you may continue.

- Ensure that the private key file is in the directory that you have chosen to use. The following examples will be based on the RedHat RPM installation default of `/etc/httpd/conf/ssl.key`.
- Ensure that the CA−signed or self−signed certificate is in its designated location. Again, I will be using the RPM default of `/etc/httpd/conf/ssl.crt`. If it is not there already, put it there.
- If there is an intermediate (root) certificate to be installed, copy it to the `/etc/httpd/conf/ssl.crt` directory, also.
- Now, you will be required to edit the httpd.conf file. Make a back−up of this file before you proceed to the next step, Configuring your Apache Server.

---

# 4. Configuring your Apache Server

The Apache server must be configured with supplementary API modules in order to support SSL. There are many SSL software packages available. My examples are based on Apache configured with ModSSL and OpenSSL. There are countless mailing lists and newsgroups available to support these products. You may find these instructions helpful for some commercial SSL software packages that are based on the Apache web server.

A few things to keep in mind: You can have multiple virtual hosts on the same server. You can have numerous name−based virtual hosts on the same IP address. You can also have numerous name−based virtual hosts and one (1) secure virtual host on the same IP. But − you cannot have multiple secure virtual hosts on the same IP. The question that so many ask: Why? The answer is: SSL works below the application layer. Name based hosts are not defined until the application layer.

Specifically, you cannot have multiple secure virtual hosts on the same SOCKET (IP address + port). By default, a secure host will use port 443. You can change configure your virtual host to use a different port number with the same IP, thus creating another socket. There are many disadvantages to this approach. The most obvious disadvantage is that if you are not using the default port, your URL must also contain the port number to access the secure site.

Example:

- Site using default port − www.something.com − would be accessed as `https://www.something.com`
- A site using port 8888 would be accessed as `https://www.something.com:8888`

Another disadvantage is that if you introduce more ports, you will be providing more opportunities for port sniffing hackers. Last, if you select a port that is used by something else, you will create conflict problem.

## 4.1 Define a Secure Virtual Host

Setting up virtual hosts is fairly straightforward. I will go through the basics of setting up a secure virtual host.

In these examples, I use the `.crt` and `.key` file extensions. That is my personal way of avoiding confusion with the various files. With Apache, you can use any extension you choose − or no extension at all.

All of your secure virtual hosts should be contained within `<IfDefine SSL>` and `</IfDefine SSL>`, usually located towards the end of the httpd.conf file.

An example of a secure virtual host:

```
<VirtualHost 172.18.116.42:443>
DocumentRoot /etc/httpd/htdocs
ServerName www.somewhere.com
ServerAdmin someone@somewhere.com
ErrorLog /etc/httpd/logs/error_log
TransferLog /etc/httpd/logs/access_log
SSLEngine on
SSLCertificateFile /etc/httpd/conf/ssl.crt/server.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl.key/server.key
SSLCACertificateFile /etc/httpd/conf/ssl.crt/ca−bundle.crt
<Files ~ "\.(cgi|shtml)$">
      SSLOptions +StdEnvVars
</Files>
<Directory "/etc/httpd/cgi−bin">
      SSLOptions +StdEnvVars
</Directory>
SetEnvIf User−Agent ".*MSIE.*" nokeepalive ssl−unclean−shutdown
CustomLog /etc/httpd/logs/ssl_request_log \
          "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
```

The directives that are the most important for SSL are the `SSLEngine on`, `SSLCertificateFile`, `SSLCertificateKeyFile`, and in many cases `SSLCACertificateFile` directives.

## SSL Engine

"`SSLEngine on`" − this is ModSSL's command to start SSL.

## SSLCertificateFile

`SSLCertificateFile` Tells Apache where to find the certificate file and what it is named. The example above shows "server.crt" as the certificate file name. This is the default that is added when you configure ModSSL with Apache. I personally don't recommend using the default names. Save yourself some frustration and name your certificates as servername.crt (domainname.crt). You may also decide to use an alternative directory than the default `/etc/httpd/conf/ssl.crt` or `/usr/local/apache/conf/ssl.crt`. Just remember to make the necessary changes to the path.

## SSLCertificateKeyFile

`SSLCertificateKeyFile` tells Apache the name of the private key and where to find it. The directory defined here should have read/write permissions for root only. No one else should have access to this directory.

## SSLCACertificateFile

The `SSLCACertificateFile` directive tells Apache where to find the Intermediate (root) certificate. This directive may or may not be necessary depending on the CA that you are using. This certificate is essentially a ring of trust.

**Intermediate Certificate** − A Certificate Authority obtains a certificate in much the same way as you. This is known as an intermediate certificate. It basically says that the holder of the intermediate certificate is whom they say they are and is authorized to issue certificates to customers. Web browsers have a list of "trusted" certificate authorities that is updated with each release. If a Certificate authority is fairly new, its intermediate certificate may not be in the browser's list of trusted CA's. Combine this with the fact that most people don't update their browsers very often; it could take years before a CA is recognized as trusted automatically. The solution is to install the intermediate certificate on the server using the `SSLCACertificateFile` directive. Usually, a "trusted" CA issues the intermediate certificate. If it is not, then you may need to use the `SSLCertificateChainFile` directive, although this is unlikely.

# 4.2 Certificate Examples

## Server Certificate File

```
-----BEGIN CERTIFICATE-----
MIIC8DCCAlmgAwIBAgIBEDANBgkqhkiG9w0BAQQFADCBxDELMAkGA1UEBhMCWkEx
FTATBgNVBAgTDFdlc3Rlcm4gQ2FwZTESMBAGA1UEBxMJQ2FwZSBUb3duMR0wGwYD
VQQKExRUaGF3dGUgQ29uc3VsdGluZyBjYzEoMCYGA1UECxMfQ2VydGlmaWNhdGlv
biBTZXJ2aWNlcyBEaXZpc2lvbjEZMBcGA1UEAxMQVGhhd3RlIFNlcnZlciBDQTEm
MCQGCSqGSIb3DQEJARYXc2VydmVyLWNlcnRzQHRoYXd0ZS5jb20wHhcNOTkwNTI1
MDMwMDAwWhcNMDIwNjEwMDMwMDAwWjBTMQswCQYDVQQGEwJVUzEbMBkGA1UEChMS
RXF1aWZheCBTZWN1cmUgSW5jMScwJQYDVQQDEx5FcXVpZmF4IFNlY3VyZSBFLUJ1
c2luZXNzIENBLTIwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAMYna8GjS9mG
q4Cb8L0VwDBMZ+ztPI05urQb8F0t1Dp4I3gOFUs2WZJJv9Y1zCFwQbQbfJuBuXmZ
QKIZJOw3jwPbfcvoTyqQhM0Yyb1YzgM2ghuv8Zz/+LYrjBo2yrmf86zvMhDVOD7z
dhDzyTxCh5F6+K6Mcmmar+ncFMmIum2bAgMBAAGjYjBgMBIGA1UdEwEB/wQIMAYB
Af8CAQAwSgYDVR0lBEMwQQYIKwYBBQUHAwEGCCsGAQUFBwMDBgorBgEEAYI3CgMD
BglghkgBhvhCBAEGCCsGAQUFBwMIBgorBgEEAYI3CgMCMA0GCSqGSIb3DQEBBAUA
A4GBALIfbC0RQ9g4Zxf/Y8IA2jWm8Tt+jvFWPt5wT3n5k0orRAvbmTROVPHGSLw7
oMNeapH1eRG5yn+erwqYazcoFXJ6AsIC5WUjAnClsSrHBCAnEn6rDU080F38xIQ3
j1FBvwMOxAq/JR5eZZcBHlSpJad88Twfd7E+0fQcqgk+nnjH
-----END CERTIFICATE-----
```

## Contents of the Certificate File

```
Certificate:
   Data:
     Version: 3 (0x2)
     Serial Number: 1516 (0x5ec)
     Signature Algorithm: md5WithRSAEncryption
     Issuer: C=US, O=Equifax Secure Inc, CN=Equifax Secure E-Business CA
     Validity
       Not Before: Jul 12 15:21:01 2000 GMT
       Not After : Jun  2 22:42:34 2001 GMT
     Subject: C=us, ST=ga, L=atlanta, O=Equifax, OU=Rick, CN=172.18.116.44/Email=richard.s
     Subject Public Key Info:
       Public Key Algorithm: rsaEncryption
       RSA Public Key: (1024 bit)
           Modulus (1024 bit):
             00:c8:eb:93:26:97:ca:00:ce:4c:e4:f3:fd:43:31:
             cd:53:ed:b4:8a:ad:93:84:dc:7a:48:39:b5:28:57:
             03:7f:a9:ac:3e:58:6a:7a:e3:52:3e:1e:52:58:a2:
             6f:23:ad:bb:84:d8:88:ed:6d:a5:da:08:6b:c8:6c:
             a5:4c:34:67:d8:46:1c:ca:20:50:b0:e8:54:7f:ca:
             5e:ef:09:ff:6e:8d:a6:2b:02:f5:54:0f:c2:d0:45:
             12:ad:66:e7:8b:dd:68:be:64:a4:9b:69:bd:a4:1a:
             5e:ef:09:ff:6e:8d:a6:2b:02:f5:54:0f:c2:d0:45:
             12:ad:66:e7:8b:dd:68:be:64:a4:9b:69:bd:a4:1a:
             5a:2f:3b:6e:73:84:d8:d6:17:bd:12:39:34:fa:3d:
             d8:a9:e8:59:3c:c2:61:c5:b3
           Exponent: 65537 (0x10001)
     X509v3 extensions:
       X509v3 Key Usage: critical
          Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment
       Netscape Cert Type:
          SSL Server
       X509v3 Authority Key Identifier:
          keyid:5B:E0:A8:75:1C:78:02:47:71:AB:CE:27:32:E7:24:88:42:28:48:56
   Signature Algorithm: md5WithRSAEncryption
     87:53:74:e9:e1:a6:10:56:8c:fa:63:0e:7b:72:ff:76:4b:79:
     0e:49:2a:58:ed:71:7a:bf:77:61:fa:e8:74:04:37:8c:d3:6a:
     9a:3d:80:76:7a:c3:64:30:e7:1b:40:25:4e:2a:81:8b:e5:ac:
     76:a4:38:67:cc:3f:93:43:e1:1d:c3:8d:ba:ed:cc:d7:aa:a4:
     ab:d3:84:77:7c:8f:26:f6:dd:ba:3b:6a:99:81:e1:9e:7e:0f:
     ca:a6:ff:c0:c3:59:6e:dc:a6:03:23:bf:8f:24:ff:15:ad:ac:
     0d:85:fc:38:bf:d1:24:2d:1a:d3:72:55:12:95:5f:65:f0:60:
     df:b1
```

## Private Key File

```
−−−−−BEGIN RSA PRIVATE KEY−−−−−
Proc−Type: 4,ENCRYPTED
DEK−Info: DES−EDE3−CBC,124F61450D85A480

ELz64SV+tFSRybsHjY9NH7CP7yDHXP6xcd9FY6MVgQykTkq2h0n7j+tmpfUPbStT
6jCgm/dTYM9mpkQ3jYZBALiVD5JNJ9t1dWisxQXY/nsak8LSTN7LhUtZSfk5xSmV
Zsl4gwQS20UdBzFiJ+4qDajP/pzocSdSuQvxIHq7UzNwJsW8UYxR3I1qrDgyNXKS
db41BWH4QdNtE0p+pi9VndDzXktqZGHEvtrQTV+39DV/dwOdnGBpYBETljMO5X6t
D42xcVs0Doa1vZ6PiMCkwFNPXsPlKHZtHwEL4I3CQdiH4E0oYh3klBzlXBY4YldN
A+s4xU44FpXp5xwt9nnVPUKHPo+NpdaRK7dAcRNO3GN3+ek1ggzvEjjuWKes3RQh
PlHPuF7VWo4KeaTfTIwJWfGxz4nvwlVByPJ6Z73Mn0VcDXCkVm6+h3PLlYL0FMqM
baUyQPpw6bhfW71FO/IIQxz3R1EqkxW7OHv74uuYl8kjHXf3S6qRZEGUG/zOGLGr
mI5s2qnU69HlBObFkc6WQq0QxMq4PiUi7HhCLMkH8+wBsNNMnb75+7lQKkEhdOeE
iUMKe5kgQqfd9w8jsBH5nu+J/nCfvPdp0isQW+P3/Rrh6YMwdKnlVfNZWdGiTzpQ
ngThAGq5lit4uf4zdTIYYrs+T9I5ltjj0KgCUD4VL5/7OfnR3gcphpbHXQf0E2cz
Qwq7q7ppKwCf/x92pHi8oVevlV5Dx9NQbGhEOA5pooqD6S2xZBbPLzkUKWDEO2il
oBZ5L1jClR5jjdF2U61w7aRrL0t6luDU/aRv/fcoYes=
−−−−−END RSA PRIVATE KEY−−−−−
```

## Contents of the Private Key

```
read RSA key
Enter PEM pass phrase:
Private−Key: (1024 bit)
modulus:
    00:c8:eb:93:26:97:ca:00:ce:4c:e4:f3:fd:43:31:
    cd:53:ed:b4:8a:ad:93:84:dc:7a:48:39:b5:28:57:
    03:7f:a9:ac:3e:58:6a:7a:e3:52:3e:1e:52:58:a2:
    6f:23:ad:bb:84:d8:88:ed:6d:a5:da:08:6b:c8:6c:
```

```
    a5:4c:34:67:d8:46:1c:ca:20:50:b0:e8:54:7f:ca:
    5e:ef:09:ff:6e:8d:a6:2b:02:f5:54:0f:c2:d0:45:
    12:ad:66:e7:8b:dd:68:be:64:a4:9b:69:bd:a4:1a:
    5a:2f:3b:6e:73:84:d8:d6:17:bd:12:39:34:fa:3d:
    d8:a9:e8:59:3c:c2:61:c5:b3
publicExponent: 65537 (0x10001)
privateExponent:
    00:b6:57:7d:3b:58:24:1e:a9:1b:85:e9:9c:9e:5f:
    d3:3d:69:0c:21:93:37:bf:2b:2c:da:e1:6c:74:48:
    cb:c7:0f:60:5f:50:74:8a:44:45:be:54:5c:5d:4e:
    45:58:f6:f1:a8:b5:af:46:f2:ec:c2:bc:43:bd:28:
    44:b7:ad:13:d3:ca:de:59:24:e8:fa:f8:e5:5f:45:
    38:2c:a0:a3:de:98:13:d8:80:38:e1:47:53:4c:ea:
    e4:66:c3:82:93:89:c3:90:83:44:e1:13:4f:74:76:
    e2:c0:89:97:77:5f:33:d8:7d:27:21:52:55:c2:d7:
    dc:01:f9:bc:21:8d:a3:f5:c1
prime1:
    00:e3:2d:6b:5e:05:6b:e1:46:e6:ab:ae:f3:8b:d0:
    5f:94:5c:6f:f5:47:46:1d:4e:66:d3:7e:98:18:e0:
    2c:0d:08:ca:b7:29:72:af:53:62:30:ec:be:26:1f:
    cc:5a:ed:65:62:65:70:1e:18:19:61:e3:77:00:a7:
    3a:9e:4e:12:93
prime2:
    00:e2:69:56:78:e8:39:ff:17:db:cc:39:d7:7f:70:
    41:dc:c5:59:43:16:c1:84:4c:ae:e7:5d:8a:c5:4b:
    da:88:8e:03:99:7c:88:f2:8a:13:31:57:44:e0:b5:
    c8:0a:60:b0:05:de:f6:9e:f2:00:ec:37:21:8d:3b:
    dc:8e:c9:d4:61
exponent1:
    1a:ad:6a:be:4f:c4:ab:5f:b8:16:d1:24:a8:76:7f:
    c2:dc:58:09:65:a5:46:2b:be:c7:77:46:45:25:8e:
    06:b9:d1:94:50:b9:b6:fd:03:ba:db:12:39:47:e2:
    a7:8a:d9:2d:04:dc:75:ac:3e:ce:cf:f7:59:8c:49:
    c5:ed:45:21
exponent2:
    2d:4e:fd:32:06:ef:0c:40:7f:08:d8:8e:6a:7f:51:
    7e:d7:b3:6c:3c:92:8f:62:35:22:31:d3:02:76:92:
    8d:ff:35:73:32:bb:c9:25:9e:7f:a2:42:33:61:cd:
    5d:5e:49:fb:72:ca:11:b6:c6:3e:7f:2d:e4:b0:95:
    0b:b2:12:21
coefficient:
    50:52:09:22:cb:fb:b2:b8:58:85:ab:1d:82:b9:6e:
    d0:f6:dc:e8:ce:a6:5d:a1:ff:c8:4d:3b:2b:1c:19:
    64:f0:c4:4a:bc:b2:1d:2b:2d:09:59:83:a3:9a:89:
    f8:db:2c:2c:8a:bd:fd:a3:16:51:76:aa:ce:ea:85:
    6b:1c:9f:f7
```

## 4.3 Restart the Web Server

The script to restart the webserver may be located in `/usr/local/sbin`, `/usr/sbin`, (where the script is called `httpd`) or `/usr/local/apache/bin` (where the script is called `apachectl`). If you are not running the server with SSL enabled, you will need to stop and start the server. You may also write your own customized scripts to start, restart, and stop your server. As long as it starts the SSL engine, you should be OK.

The commands are:

```
httpd stop
httpd startssl
httpd restart
```

*or*

```
apachectl stop
apachectl startssl
apachectl restart
```

---

# 5. Troubleshooting

Here are some common problems that may arise.

## 5.1 Server Appears to start, but you cannot access the secure site

Check the `error_log` file. If you did not set your virtual host to write to an error log, you may want to reconsider. The example SSL virtual host writes to an error log file. Most likely you will have a few warnings and an error at the end of the log that basically say that the private key does not match the certificate.

Example:

```
[Tue Nov 21 09:09:02 2000] [notice] Apache/1.3.14 (Unix) mod_ssl/2.7.1
OpenSSL/0.9.6 configured −− resuming normal operations
[Tue Nov 21 09:09:16 2000] [notice] caught SIGTERM, shutting down
[Tue Nov 21 14:39:54 2000] [notice] Apache/1.3.14 (Unix) mod_ssl/2.7.1
OpenSSL/0.9.6 configured −− resuming normal operations
[Tue Nov 21 14:40:31 2000] [notice] caught SIGTERM, shutting down
[Tue Nov 21 14:43:53 2000] [error] mod_ssl: Init: (esi.fin.equifax.com:443)
Unable to configure RSA server private key (OpenSSL library error follows)
[Tue Nov 21 14:43:53 2000] [error] OpenSSL: error:0B080074:x509 certificate
routines:X509_check_private_key:key values mismatch
```

If you get the error messages above, chances are the key and certificate do not match. Make sure you aren't using the default `server.key` file. You should also check the `httpd.conf` file to make sure that the directives are pointing to the correct private key and certificate.

You can check to make sure that you your private key and certificate are in the correct format and match each other. To do this, give the commands below to decrypt the private key in one terminal window and decrypt the certificate in the other. What you will be comparing are the Modulus and the Exponent of each key. If the modulus and exponent from the key matches the set from the certificate, you have just confirmed that your certificate and key are correctly paired.

If all else fails, create a new private key, CSR or self−signed certificate. Before you do this, check your CA's re−issue policy. You may be charged for a re−issue.

To view the contents of the certificate:

```
openssl x509 −noout −text −in filename.crt
```

To view the contents of the private key:

```
openssl rsa −noout −text −in filename.key
```

## 5.2 Certificate Name Check Warning is issued by the client's browser

The most common cause for this is omitting the "www" at the beginning of the domain name when creating the CSR. The name defined by the "ServerName" directive for that virtual host must match the domain name presented by the certificate exactly or the browser will let the client know. The exception is a wild card certificate. A wild card certificate's domain name field would look like *.somedomain.com. This enables you to use one certificate for any number of sub−domains of somedomain.com (e.g. host1.somedomain.com and host2.somedomain.com).

## 5.3 Certificate was Signed by an Untrusted Certificate Authority Warning is issued by the client's browser

If you are using a self−signed certificate, you will get this warning. Your clients will be given the option to trust your certificate or not. If you have a CA signed certificate and are getting the untrusted warning, you probably need to install their intermediate (root) certificate.

## 5.4 SSLEngine on is an un−recognized command (when starting Apache)

This error message is issued if you do not have ModSSL compiled with Apache. Some SSL packages use a different directive to start SSL within a virtual host. If you are using a package that does use a different directive, you will also receive this error message.

## 5.5 You have forgotten your "PEM Passphrase" and you would like to know how to reset it

There is no way to reset this passphrase. The only solution is to remember the passphrase or create a new private key. You will then need to obtain a new certificate or create a new self−signed certificate.

---

# 6. Glossary

***Authentication***

The positive identification of a network entity such as a server, a client, or a user. In SSL context, authentication represents the server and client Certificate verification process.

***Access Control***

The restriction of access to network realms. In Apache context usually the restriction of access to certain URLs.

***Algorithm***

An unambiguous formula or set of rules for solving a problem in a finite number of steps. Algorithms for encryption are usually called Ciphers.

***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 Authority (CA)***

A trusted third party whose purpose is to sign certificates for network entities that 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.

***Ciphertext***

The result after a Plaintext passed a Cipher.

***Configuration Directive***

A configuration command that controls one or more aspects of a program's behavior. In Apache context these are all the command names in the first column of the configuration files.

***Cryptography − Symmetric***

The client and server use the same key to encrypt and to decrypt data.

***Cryptography − Asymmetric***

Consists of a key pair (public and private). PKI is Asymmetric Cryptography

***Digital Signatures***

A piece of data that is sent with an encrypted message that identifies the originator and verifies that it has not been altered.

***HTTPS***

The HyperText Transport Protocol (Secure), the standard encrypted communication mechanism on the World Wide Web. This is actually just HTTP over SSL.

***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.

***Non−repudiation***

A service that provides proof of the integrity and origin of data, both in an non−forgeable relationship, which can be verified by any third party at any time, or, an authentication that with high assurance can be asserted to be genuine.

A property achieved through cryptographic methods which prevents an individual or entity from denying having performed a particular action related to data (such as mechanisms for non−rejection or authority (origin); for proof of obligation, intent, or commitment, or for proof of ownership).

***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.

***Plaintext***

The unencrypted text.

***Private Key***

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

***Public Key***

The publicly 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 Cryptography.

***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, i.e. the HyperText Transfer Protocol (HTTP) over SSL.

***Session***

The context information of an SSL communication.

***SSLeay***

The original SSL/TLS implementation library developed by Eric A. Young <eay@aus.rsa.com>; see 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.

***Uniform Resource Locator (URL)***

The formal identifier to locate various resources on the World Wide Web. The most popular URL scheme is http. SSL uses the scheme https

***X.509***

An authentication certificate scheme recommended by the International Telecommunication Union (ITU−T) and used for SSL/TLS authentication.

***ITU−T***

Recommendation X.509 [CCI88c] specifies the authentication service for X.500 directories, as well as the X.509 certificate syntax. Directory authentication in X.509 can be carried out using either secret−key techniques or public−key techniques; the latter is based on public−key certificates. The standard does not specify a particular cryptographic algorithm, although an informative annex of the standard describes the RSA algorithm.