How To Create an SSL Certificate on Apache for CentOS 7

## **Prerequisites**

**1**. You will also need to have Apache installed in order to configure virtual hosts for it. If you haven’t already done so, you can use yum to install Apache through CentOS’s default software repositories:

**sudo yum install httpd**

**2**. Next, enable Apache as a CentOS service so that it will automatically start after a reboot:

**sudo systemctl enable httpd.service**

## **Step One: Install Mod SSL**

**1**. In order to set up the self-signed certificate, we first have to be sure that mod\_ssl, an Apache module that provides support for SSL encryption, is installed the server. We can install mod\_ssl with the yum command:

**sudo yum install mod\_ssl**

The module will automatically be enabled during installation, and Apache will be able to start using an SSL certificate after it is restarted. You don’t need to take any additional steps for mod\_ssl to be ready for use.

## **Step Two: Create a New Certificate**

**1**. Now that Apache is ready to use encryption, we can move on to generating a new SSL certificate. The certificate will store some basic information about your site, and will be accompanied by a key file that allows the server to securely handle encrypted data.

First, we need to create a new directory to store our private key (the /etc/ssl/certs directory is already available to hold our certificate file):

**sudo mkdir /etc/ssl/private**

**2**. Since files kept within this directory must be kept strictly private, we will modify the permissions to make sure only the root user has access:

**sudo chmod 700 /etc/ssl/private**

**3**. Now that we have a location to place our files, we can create the SSL key and certificate files with openssl:

**sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt**

After you enter the request, you will be taken to a prompt where you can enter information about your website.

**4**. While we are using OpenSSL, we should also create a strong Diffie-Hellman group, which is used in negotiating [Perfect Forward Secrecy](https://en.wikipedia.org/wiki/Forward_secrecy) with clients.

We can do this by typing:

**sudo openssl dhparam -out /etc/ssl/certs/dhparam.pem 2048**

**5**. Since the version of Apache that ships with CentOS 7 does not include the SSLOpenSSLConfCmd directive, we will have to manually append the generated file to the end of our self-signed certificate. To do this, type:

**cat /etc/ssl/certs/dhparam.pem | sudo tee -a /etc/ssl/certs/apache-selfsigned.crt**

## **Step Three: Set Up the Certificate**

**1**. We now have all of the required components of the finished interface. The next thing to do is to set up the virtual hosts to display the new certificate.

Open Apache’s SSL configuration file in your text editor with root privileges:

**sudo vi /etc/httpd/conf.d/ssl.conf**

**2**. Adjusting the VirtualHost Directives

First, uncomment the DocumentRoot line and edit the address in quotes to the location of your site’s document root. By default, this will be in /var/www/html, and you don’t need to change this line if you have not changed the document root for your site. However, if you followed a guide like our [Apache virtual hosts setup guide](https://www.digitalocean.com/community/tutorials/how-to-set-up-apache-virtual-hosts-on-centos-7), your site’s document root may be different.

Next, uncomment the ServerName line and replace www.example.com with your domain name or server IP address (whichever one you put as the common name in your certificate):

/etc/httpd/conf.d/ssl.conf

<VirtualHost \_default\_:443>

. . .

DocumentRoot "/var/www/example.com/public\_html"

ServerName www.example.com:443

Next, find the SSLProtocol and SSLCipherSuite lines and either delete them or comment them out. The configuration we be pasting in a moment will offer more secure settings than the default included with CentOS’s Apache:

/etc/httpd/conf.d/ssl.conf

. . .

# SSLProtocol all -SSLv2

. . .

# SSLCipherSuite HIGH:MEDIUM:!aNULL:!MD5:!SEED:!IDEA

Find the SSLCertificateFile and SSLCertificateKeyFile lines and change them to the directory we made at /etc/httpd/ssl:

/etc/httpd/conf.d/ssl.conf

SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt

SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

We’re now done with the changes within the actual VirtualHost block. The next changes will take place after the ending </VirtualHost> tag within this same file.

Paste in the settings from the site **AFTER** the end of the VirtualHost block:

/etc/httpd/conf.d/ssl.conf

. . .

</VirtualHost>

. . .

# Begin copied text

# from https://cipherli.st/

# and https://raymii.org/s/tutorials/Strong\_SSL\_Security\_On\_Apache2.html

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH

SSLProtocol All -SSLv2 -SSLv3

SSLHonorCipherOrder On

# Disable preloading HSTS for now. You can use the commented out header line that includes

# the "preload" directive if you understand the implications.

#Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains; preload"

Header always set Strict-Transport-Security "max-age=63072000; includeSubdomains"

Header always set X-Frame-Options DENY

Header always set X-Content-Type-Options nosniff

# Requires Apache >= 2.4

SSLCompression off

SSLUseStapling on

SSLStaplingCache "shmcb:logs/stapling-cache(150000)"

# Requires Apache >= 2.4.11

# SSLSessionTickets Off

**3**. To redirect all traffic to be SSL encrypted, create and open a file ending in .conf in the /etc/httpd/conf.d directory:

**sudo vi /etc/httpd/conf.d/non-ssl.conf**

Inside, create a VirtualHost block to match requests on port 80. Inside, use the ServerName directive to again match your domain name or IP address. Then, use Redirect to match any requests and send them to the SSL VirtualHost. Make sure to include the trailing slash:

/etc/apache2/sites-available/000-default.conf

<VirtualHost \*:80>

ServerName www.example.com

Redirect "/" "https://www.example.com/"

</VirtualHost>

## **Step Four: Activate the Certificate**

**1**. By now, you have created an SSL certificate and configured your web server to apply it to your site. To apply all of these changes and start using your SSL encryption, you can restart the Apache server to reload its configurations and modules.

First, check your configuration file for syntax errors by typing:

**sudo apachectl configtest**

Next, make sure port 80 and 443 are open in your firewall. If you are not running a firewall, you can skip ahead.

If you have a **firewalld** firewall running, you can open these ports by typing:

**sudo firewall-cmd --add-service=http**

**sudo firewall-cmd --add-service=https**

**sudo firewall-cmd --runtime-to-permanent**

If have an **iptables** firewall running, the commands you need to run are highly dependent on your current rule set. For a basic rule set, you can add HTTP and HTTPS access by typing:

**sudo iptables -I INPUT -p tcp -m tcp --dport 80 -j ACCEPT**

**sudo iptables -I INPUT -p tcp -m tcp --dport 443 -j ACCEPT**

**In your web browser, try visiting your domain name :**

**https://example.com/**

# How To Create a Self-Signed SSL Certificate for Apache in Ubuntu 18.04

## **Step 1 – Creating the SSL Certificate**

TLS/SSL works by using a combination of a public certificate and a private key. The SSL key is kept secret on the server. It is used to encrypt content sent to clients. The SSL certificate is publicly shared with anyone requesting the content. It can be used to decrypt the content signed by the associated SSL key.

We can create a self-signed key and certificate pair with OpenSSL in a single command:

* **sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout /etc/ssl/private/apache-selfsigned.key -out /etc/ssl/certs/apache-selfsigned.crt**

As we stated above, these options will create both a key file and a certificate. We will be asked a few questions about our server in order to embed the information correctly in the certificate.

## **Step 2 – Configuring Apache to Use SSL**

We have created our key and certificate files under the /etc/ssl directory. Now we just need to modify our Apache configuration to take advantage of these.

First, we will create an Apache configuration snippet to define some SSL settings. This will set Apache up with a strong SSL cipher suite and enable some advanced features that will help keep our server secure. The parameters we will set can be used by any Virtual Hosts enabling SSL.

Create a new snippet in the /etc/apache2/conf-available directory. We will name the file ssl-params.conf to make its purpose clear:

**sudo nano /etc/apache2/conf-available/ssl-params.conf**

Paste the configuration into the ssl-params.conf file we opened:

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH

SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1

SSLHonorCipherOrder On

# Disable preloading HSTS for now. You can use the commented out header line that includes

# the "preload" directive if you understand the implications.

# Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"

Header always set X-Frame-Options DENY

Header always set X-Content-Type-Options nosniff

# Requires Apache >= 2.4

SSLCompression off

SSLUseStapling on

SSLStaplingCache "shmcb:logs/stapling-cache(150000)"

# Requires Apache >= 2.4.11

SSLSessionTickets Off

Save and close the file when you are finished.

### **Modifying the Default Apache SSL Virtual Host File**

Next, let’s modify /etc/apache2/sites-available/default-ssl.conf, the default Apache SSL Virtual Host file. If you are using a different server block file, substitute its name in the commands below.

Before we go any further, let’s back up the original SSL Virtual Host file:

* **sudo cp /etc/apache2/sites-available/default-ssl.conf /etc/apache2/sites-available/default-ssl.conf.bak**

Now, open the SSL Virtual Host file to make adjustments:

* **sudo nano /etc/apache2/sites-available/default-ssl.conf**

We will be making some minor adjustments to the file. We will set the normal things we’d want to adjust in a Virtual Host file (ServerAdmin email address, ServerName, etc., and adjust the SSL directives to point to our certificate and key files.

After making these changes, your server block should look similar to this:

/etc/apache2/sites-available/default-ssl.conf

<IfModule mod\_ssl.c>

<VirtualHost \_default\_:443>

ServerAdmin your\_email@example.com

ServerName server\_domain\_or\_IP

DocumentRoot /var/www/html

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

SSLEngine on

SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt

SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key

<FilesMatch "\.(cgi|shtml|phtml|php)$">

SSLOptions +StdEnvVars

</FilesMatch>

<Directory /usr/lib/cgi-bin>

SSLOptions +StdEnvVars

</Directory>

</VirtualHost>

</IfModule>

Save and close the file when you are finished.

### **Modifying the HTTP Host File to Redirect to HTTPS**

To adjust the unencrypted Virtual Host file to redirect all traffic to be SSL encrypted, we can open the /etc/apache2/sites-available/000-default.conf file:

**sudo nano /etc/apache2/sites-available/000-default.conf**

Inside, within the VirtualHost configuration blocks, we need to add a Redirect directive, pointing all traffic to the SSL version of the site:

/etc/apache2/sites-available/000-default.conf

<VirtualHost \*:80>

. . .

Redirect "/" "https://your\_domain\_or\_IP/"

. . .

</VirtualHost>

## **Step 3 – Adjusting the Firewall**

We can see the available profiles by typing:

**sudo ufw app list**

Available applications:

Apache

Apache Full

Apache Secure

OpenSSH

To additionally let in HTTPS traffic, we can allow the “Apache Full” profile and then delete the redundant “Apache” profile allowance:

* **sudo ufw allow 'Apache Full'**
* **sudo ufw delete allow 'Apache'**

## **Step 4 – Enabling the Changes in Apache**

We can enable mod\_ssl, the Apache SSL module, and mod\_headers, needed by some of the settings in our SSL snippet, with the a2enmod command:

* **sudo a2enmod ssl**
* **sudo a2enmod headers**

Next, we can enable our SSL Virtual Host with the a2ensite command:

**sudo a2ensite default-ssl**

We will also need to enable our ssl-params.conf file, to read in the values we set:

**sudo a2enconf ssl-params**

At this point, our site and the necessary modules are enabled. We should check to make sure that there are no syntax errors in our files. We can do this by typing:

**sudo apache2ctl configtest**

AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress this message

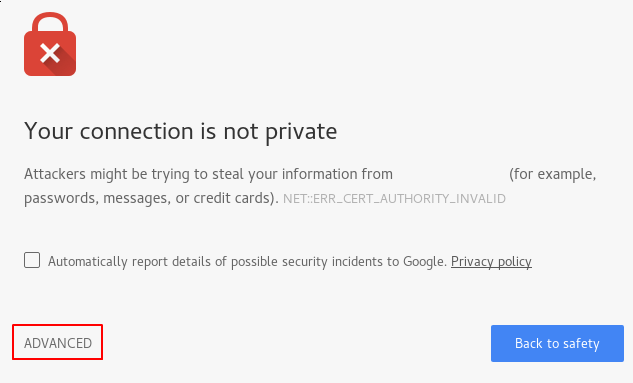
Syntax OK

## **Step 5 – Testing Encryption**

Open your web browser and type https:// followed by your server’s domain name or IP into the address bar:

https://server\_domain\_or\_IP

Because the certificate we created isn’t signed by one of your browser’s trusted certificate authorities, you will likely see a scary looking warning like the one below:



This is expected and normal.