

vps8986.vpsunit.com

mail.shopsvee.com
mail.workvee.com
mail.filevee.com
mail.shopsvee.com
mail.passwordsense.com
mail.1datapoint.com
mail.regionlookupu.com
mail.careermeds.com
mail.jobench.com
mail.findthatip.com

Imagick Policy

sudo nano /etc/ImageMagick-6/policy.xml

change

```
<policy domain="coder" rights="none" pattern="PDF" />
```

to

```
<policy domain="coder" rights="read|write" pattern="PDF" />
```

```
sudo mkdir -p /var/spool/postfix/var/run/openssl
sudo chown openssl:openssl /var/spool/postfix/var/run/openssl
usermod -a -G openssl postfix
sudo service openssl restart
```

Postfix/Dovecot multi domains and multi certs on one IP

Step 1: Comment out the top two lines and add the follow lines to /etc/postfix/main.cf:

```
# smtpd_tls_cert_file = /etc/pki/dovecot/certs/dovecot.pem
# smtpd_tls_key_file = /etc/pki/dovecot/private/dovecot.pem

# provide the primary certificate for the server, to be used for outgoing
connections (note the indentation)
smtpd_tls_chain_files =
    /etc/letsencrypt/live/mail.yourprimarymailserverdomain.com/privkey.pem,
    /etc/letsencrypt/live/mail.yourprimarymailserverdomain.com/fullchain.pem

# provide the map to be used when SNI support is enabled
tls_server_sni_maps = hash:/etc/postfix/vmail_ssl.map
```

Step 2: Create the file /etc/postfix/vmail_ssl.map with the following:

```
# Compile with postmap -F hash:/etc/postfix/vmail_ssl.map when updating
# One host per line
mail.yourprimarymailserverdomain.com
/etc/letsencrypt/live/mail.yourprimarymailserverdomain.com/privkey.pem
/etc/letsencrypt/live/mail.yourprimarymailserverdomain.com/fullchain.pem
mail.yoursecondarymailserverdomain.com
/etc/letsencrypt/live/mail.yoursecondarymailserverdomain.com/privkey.pem
/etc/letsencrypt/live/mail.yoursecondarymailserverdomain.com/fullchain.pem
# add more domains with keys and certs as needed
```

Step 3: Run postmap -F hash:/etc/postfix/vmail_ssl.map.

Step 4: Run systemctl restart postfix.

Step 5: Now test your domains' SSLs! For each of your domains, run the following command:

```
openssl s_client -connect localhost:25 -servername mail.mydomainname.com -starttls smtp
```

Step 6: Dovecot

/etc/dovecot/conf.d/auth-system.conf.ext

comment below:

```
userdb {  
    driver = passwd  
}
```

/etc/dovecot/conf.d/10-ssl.conf

Default

```
ssl_cert = </path/to/default/cert  
ssl_key = </path/to/default/private/key
```

mail.example.it

```
local_name mail.example.it {  
    ssl_cert = </etc/letsencrypt/live/mail.example.it  
    ssl_key = </path/to/mail.example.it/private/key  
}
```

mail.example.com

```
local_name mail.example.com {  
    ssl_cert = </etc/letsencrypt/live/mail.example.com  
    ssl_key = </path/to/mail.example.com/private/key  
}
```

How to check if port is in use

```
sudo ss -lnpt | grep master  
sudo lsof -i -P -n | grep LISTEN  
sudo netstat -tulpn | grep LISTEN  
sudo ss -tulpn | grep LISTEN  
sudo lsof -i:22 ## see a specific port such as 22 ##  
sudo nmap -sTU -O IP-address-Here
```

Read email address from Database (Dovecot)

sudo nano /etc/dovecot/dovecot-dict-sql.conf.ext

```
* edit line = connect = host=192.168.1.1 dbname=dbname user=user  
password=password  
* add to the following files
```

```
user = db_user  
password = db_password  
hosts = db_host  
dbname = db_name  
query = SELECT 1 FROM db_table WHERE email='%s'
```

/etc/postfix/mysql-virtual-alias-maps.cf

/etc/postfix/mysql-virtual-mailbox-domains.cf

/etc/postfix/mysql-virtual-mailbox-maps.cf

PDF utilities (based on Poppler)

Poppler is a PDF rendering library based on Xpdf PDF viewer.

This package contains command line utilities (based on Poppler) for getting information of PDF documents, convert them to other formats, or manipulate them:

```
sudo apt-get install poppler-utils
* pdfdetach -- lists or extracts embedded files (attachments)
* pdffonts -- font analyzer
* pdfimages -- image extractor
* pdfinfo -- document information
* pdfseparate -- page extraction tool
* pdfsig -- verifies digital signatures
* pdftocairo -- PDF to PNG/JPEG/PDF/PS/EPS/SVG converter using Cairo
* pdftohtml -- PDF to HTML converter
* pdftoppm -- PDF to PPM/PNG/JPEG image converter
* pdftops -- PDF to PostScript (PS) converter
* pdftotext -- text extraction
* pdfunite -- document merging tool
```

Installing Certbot

```
sudo apt install certbot python3-certbot-apache
sudo certbot --apache
https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-20-04#step-5-verifying-certbot-auto-renewal
```

Installing Git on Debian

```
sudo apt install git
git --version
```

Installing Composer on Debian

```
cd ~
sudo apt install wget php-cli php-zip unzip
wget -O composer-setup.php https://getcomposer.org/installer
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
---OR---
cd ~
sudo apt install curl php-cli php-mbstring git unzip
curl -sS https://getcomposer.org/installer -o composer-setup.php
HASH=`curl -sS https://composer.github.io/installer.sig`
echo $HASH
php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo
'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-
setup.php'); } echo PHP_EOL;"
sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

Configuration

```
#install and enable resolvconf
apt install resolvconf
systemctl start resolvconf.service
systemctl enable resolvconf.service

#install network manager
sudo apt install network-manager -y

#set permanent DNS server in resolvconf
vim /etc/resolvconf/resolv.conf.d/
eg: nameserver 8.8.8.8

#install dns utils
sudo apt install dnsutils -y

#start apache service
systemctl start apache2

#open file for editing
vim /etc/apache2/conf-enabled/security.conf
ServerTokens Prod
ServerSignature Off

#restart apache service
systemctl restart apache2

#turn off directory listing
vim /etc/apache2/apache2.conf
<Directory /var/www/>
    Options -Indexes
    Options FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

#create virtualhost for single or multiple websites
cd /etc/apache2/sites-available
nano <domain.com.conf>
<VirtualHost *:80>
    ServerName domain.com
    ServerAlias www.domain.com
    DocumentRoot /var/www/domain.com
</VirtualHost>

#restart apache service
systemctl restart apache2

-----
Install static IP Address on Debian
-----
#show network interfaces
ip -c link show

#backup network interfaces
sudo cp /etc/network/interfaces ~/
or
cp -a /etc/network/interfaces /etc/network/interfaces.backup

#open for editing
sudo nano /etc/network/interfaces

#restart networking service
sudo systemctl restart networking
```

```

#view dns server
cat /etc/resolv.conf

#show default gateway
ip route

#open syslog
less /var/log/syslog

-----
Add user to sudo group
-----

#login as root
su - root

#make sure sudo is installed
apt install sudo

#add user to sudo
usermod -aG sudo <username>

-----
#Enable PHP 8.2 FPM extension
sudo a2enconf php8.2-fpm

#Disable PHP 8.1 FPM extension
sudo a2disconf php8.1-fpm

-----
Install/Update Latest PHP (8.2)

sudo apt update
sudo apt install lsb-release apt-transport-https ca-certificates software-
properties-common -y
sudo wget -O /etc/apt/trusted.gpg.d/php.gpg
https://packages.sury.org/php/apt.gpg
sudo sh -c 'echo "deb https://packages.sury.org/php/ $(lsb_release -sc) main"
> /etc/apt/sources.list.d/php.list'
sudo apt update
sudo apt install php8.2
php -v
sudo apt install php8.2-
{cli,zip,mysql,bz2,curl,mbstring,intl,common,imagick,fpm,xml,gd,ldap}
apt install php8.2-cli php8.2-xml php8.2-gd php8.2-curl php8.2-mysql php8.2-ldap
php8.2-zip php8.2-imagick -y
sudo apt install apache2 libapache2-mod-php8.2
sudo a2enmod php8.2
sudo a2enconf php8.2-fpm
sudo systemctl restart apache2

restart php
-----
/etc/init.d/php8.2-fpm restart
service php8.2-fpm restart

-----

#Remove php
sudo apt purge php8.1*

```

Firewall

#disable
sudo ufw disable

#remove
sudo apt-get remove ufw

#purge
sudo apt-get purge ufw

RoundCube X-Frame-Options

grep -ri "X-Frame-Options" /etc/apache2
/etc/apache2/conf-available/ssl-params.conf
"Header always set X-Frame-Options DENY" to "Header always set X-Frame-Options
SAMEORIGIN"
sudo service apache2 restart

RoundCube

Roundcube is a free open-source, full-featured webmail client written in PHP. A
webmail is a mail client in your browser.
Instead of reading and sending emails from a desktop mail client like Mozilla
Thunderbird, you can access your email from a web browser.
This tutorial is going to show you how to install Roundcube webmail on Debian
11/10 with Apache or Nginx web server.
Roundcube Features

Roundcube functionality includes:

- Address book
- Folder management
- Message searching
- Message filter
- Spell checking
- MIME support
- PGP encryption and signing
- Mailvelope integration
- Users are able to change their passwords in Roundcube.
- Import MIME or Mbox formatted emails.
- Email Resent (Bounce)
- Support for Redis and Memcached cache
- Support for SMTPUTF8 and GSSAPI
- A responsive skin called Elastic with full mobile device support
- OAuth2/XOAuth support (with plugin hooks)
- Collected recipients and trusted senders
- Full Unicode support with MySQL database
- Support of IMAP LITERAL- extension

Requirements

To follow this tutorial, it's assumed that

Postfix SMTP server and Dovecot IMAP server have been installed on your
Debian 11/10 server

If not, please click the above links and follow the instructions to complete the prerequisites. Note that if you set up your email server using iRedMail before, then your server meets all requirements and Roundcube is already installed on your server.

Now let's proceed to install Roundcube.

Step 1: Download Roundcube Webmail on Debian 11/10

Log in to your Debian server via SSH, then run the following command to download the latest stable version from Roundcube Github repository.

```
wget https://github.com/roundcube/roundcubemail/releases/download/1.6.0/roundcubemail-1.6.0-complete.tar.gz
```

Note: You can always use the above URL format to download Roundcube from command line. If a new version comes out, simply replace 1.6.0 with the new version number. You can check if there's new release at Roundcube download page.

Extract the tarball, move the newly created folder to web root (/var/www/) and rename it as roundcube at the same time.

```
tar xvf roundcubemail-1.6.0-complete.tar.gz
```

```
sudo mkdir -p /var/www/
```

```
sudo mv roundcubemail-1.6.0 /var/www/roundcube
```

Change into the roundcube directory.

```
cd /var/www/roundcube
```

Make the web server user (www-data) as the owner of the temp and logs directory so that web server can write to these two directories.

```
sudo chown www-data:www-data temp/ logs/ -R
```

Step 2: Install PHP Extensions

Run the following command to install the required PHP extensions.

```
sudo apt install software-properties-common
```

```
sudo add-apt-repository ppa:ondrej/php
```

```
sudo apt update
```

```
sudo apt install php-net-ldap2 php-net-ldap3 php-imagick php8.1-fpm php8.1-common php8.1-gd php8.1-imap php8.1-mysql php8.1-curl php8.1-zip php8.1-xml php8.1-mbstring php8.1-bz2 php8.1-intl php8.1-gmp php8.1-redis
```

Step 3: Create a MariaDB Database and User for Roundcube

Log into MariaDB shell as root.

```
sudo mysql -u root -h localhost -p
```

Then create a new database for Roundcube using the following command. This tutorial name it roundcube, you can use whatever name you like for the database.

```
CREATE DATABASE roundcubemail DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Next, create a new database user on localhost using the following command. Again, this tutorial name it roundcubeuser, you can use whatever name you like.

Replace password with your preferred password.

```
CREATE USER roundcube@localhost IDENTIFIED BY 'password';
```

Then grant all permission of the new database to the new user so later on Roundcube webmail can write to the database.

```
GRANT ALL PRIVILEGES ON roundcubemail.* TO roundcube@localhost;
```

Flush the privileges table for the changes to take effect.

```
flush privileges;
```

Exit MariaDB Shell:

```
exit;
```

Import the initial tables to roundcube database.

```
sudo mysql roundcubemail < /var/www/roundcube/SQL/mysql.initial.sql
```

Step 4: Create Apache Virtual Host or Nginx Config File for Roundcube Apache

If you use Apache web server, create a virtual host for Roundcube.

```
sudo nano /etc/apache2/sites-available/roundcube.conf
```

Note: If you followed my Postfix/Dovecot tutorial, a virtual host already exists. you should edit the following file. (Delete the existing content.)

```
sudo nano /etc/apache2/sites-available/mail.example.com.conf
```

Put the following text into the file. Replace mail.example.com with your real domain name and don't forget to set DNS A record for it.

```
<VirtualHost *:80>
    ServerName mail.example.com
    DocumentRoot /var/www/roundcube/

    ErrorLog ${APACHE_LOG_DIR}/roundcube_error.log
    CustomLog ${APACHE_LOG_DIR}/roundcube_access.log combined

    <Directory />
        Options FollowSymLinks
        AllowOverride All
    </Directory>

    <Directory /var/www/roundcube/>
        Options FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
</VirtualHost>
```

Save and close the file. Then enable this virtual host with:

```
sudo a2ensite roundcube.conf
```

Reload Apache for the changes to take effect.

```
sudo systemctl reload apache2
```


Now you should be able to see the Roundcube web-based install wizard at <http://mail.example.com/installer>.
Nginx

If you use Nginx web server, create a virtual host for Roundcube.

```
sudo nano /etc/nginx/conf.d/roundcube.conf
```

Note: If you followed my Postfix/Dovecot tutorial, a virtual host already exists. you should edit the following file. (Delete the existing content.)

```
sudo nano /etc/nginx/conf.d/mail.example.com.conf
```

Put the following text into the file. Replace the domain name and don't forget to set DNS A record for it.

```
server {
    listen 80;
    listen [::]:80;
    server_name mail.example.com;
    root /var/www/roundcube/;
    index index.php index.html index.htm;

    error_log /var/log/nginx/roundcube.error;
    access_log /var/log/nginx/roundcube.access;

    location / {
        try_files $uri $uri/ /index.php;
    }

    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/run/php/php8.1-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~ /\.well-known/acme-challenge {
        allow all;
    }
    location ~ ^/(README|INSTALL|LICENSE|CHANGELOG|UPGRADING)$ {
        deny all;
    }
    location ~ ^/(bin|SQL)/ {
        deny all;
    }
    # A long browser cache lifetime can speed up repeat visits to your page
    location ~* \.(jpg|jpeg|gif|png|webp|svg|woff|woff2|ttf|css|js|ico|xml)$ {
        access_log off;
        log_not_found off;
        expires 360d;
    }
}
```

Save and close the file. Then test Nginx configurations.

```
sudo nginx -t
```

If the test is successful, reload Nginx for the changes to take effect.

```
sudo systemctl reload nginx
```

Now you should be able to see the Roundcube web-based install wizard at <http://mail.example.com/installer>.
Step 5: Enabling HTTPS

It's highly recommended that you use TLS to encrypt your webmail. We can enable HTTPS by installing a free TLS certificate issued from Let's Encrypt. Run the following command to install Let's Encrypt client (certbot) on Debian 11/10 server.

```
sudo apt install certbot
```

If you use Nginx, then you also need to install the Certbot Nginx plugin.

```
sudo apt install python3-certbot-nginx
```

Next, run the following command to obtain and install TLS certificate.

```
sudo certbot --nginx --agree-tos --redirect --hsts --staple-ocsp --email  
you@example.com -d mail.example.com
```

If you use Apache, install the Certbot Apache plugin.

```
sudo apt install python3-certbot-apache
```

And run this command to obtain and install TLS certificate.

```
sudo certbot --apache --agree-tos --redirect --hsts --staple-ocsp --email  
you@example.com -d mail.example.com
```

```
#REMOVE DOVECOT  
systemctl stop dovecot  
systemctl disable dovecot  
apt-get purge dovecot-core  
apt-get autoremove dovecot-core
```

```
#PURGE AND REINSTALL POSTFIX  
apt-get --purge remove postfix  
apt purge courier-imap  
apt install courier-imap  
apt install courier-pop
```

```
sudo apt-get update && sudo apt-get upgrade  
sudo apt-get install postfix postfix-mysql dovecot-core dovecot-imapd dovecot-  
pop3d dovecot-lmtpd dovecot-mysql
```

```
sudo apt update  
sudo apt install mailutils  
sudo apt install postfix  
Select "Internet Site"  
Modify /etc/postfix/main.cf  
change from: inet_interfaces = all to inet_interfaces = loopback-only  
sudo systemctl restart postfix  
-----
```

```
/usr/local/ssl/openssl.cnf  
/usr/lib/ssl/openssl.cnf  
/etc/ssl/openssl.cnf
```

UPDATE DEBIAN SYSTEM

```
-----  
apt-get update  
apt-get upgrade
```

```
INSTALL APACHE2
```

```
-----  
sudo apt install apache2
```

```
#ENABLE SITE
```

```
-----  
a2ensite /etc/apache2/sites-available/{example.com}.conf
```

```
#DISABLE SITE
```

```
-----  
a2disssite /etc/apache2/sites-available/{example.com}.conf
```

```
-----  
How To Create a Self-Signed SSL Certificate for Apache in Debian 10
```

Published on July 22, 2019

Apache

Security

Debian 10

Default avatar

By Brian Boucheron, Mark Drake, and Erika Heidi

English

How To Create a Self-Signed SSL Certificate for Apache in Debian 10

Not using Debian 10?

Choose a different version or distribution.

Debian 10

Introduction

TLS, or transport layer security, and its predecessor SSL, which stands for secure sockets layer, are web protocols used to wrap normal traffic in a protected, encrypted wrapper.

Using this technology, servers can send traffic safely between servers and clients without the possibility of messages being intercepted by outside parties. The certificate system also assists users in verifying the identity of the sites that they are connecting with.

In this guide, we will show you how to set up a self-signed SSL certificate for use with an Apache web server on Debian 10.

Note: A self-signed certificate will encrypt communication between your server and any clients. However, because it is not signed by any of the trusted certificate authorities included with web browsers, users cannot use the certificate to validate the identity of your server automatically.

A self-signed certificate may be appropriate if you do not have a domain name associated with your server and for instances where an encrypted web interface is not user-facing. If you do have a domain name, in many cases it is better to use a CA-signed certificate. You can find out how to set up a free trusted certificate with the Let's Encrypt project [here](#).

Prerequisites

Before you begin, you should have a non-root user configured with sudo privileges. You can learn how to set up such a user account by following our Initial Server Setup with Debian 10.

You will also need to have the Apache web server installed. If you would like to install an entire LAMP (Linux, Apache, MariaDB, PHP) stack on your server, you can follow our guide on setting up LAMP on Debian 10. If you just want the Apache web server, skip the steps pertaining to PHP and MariaDB.

When you have completed these prerequisites, continue below.

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
```

You will be asked a series of questions. Before we go over that, let's take a look at what is happening in the command we are issuing:

openssl: This is the basic command line tool for creating and managing OpenSSL certificates, keys, and other files.

req: This subcommand specifies that we want to use X.509 certificate signing request (CSR) management. The "X.509" is a public key infrastructure standard that SSL and TLS adheres to for its key and certificate management. We want to create a new X.509 cert, so we are using this subcommand.

-x509: This further modifies the previous subcommand by telling the utility that we want to make a self-signed certificate instead of generating a certificate signing request, as would normally happen.

-nodes: This tells OpenSSL to skip the option to secure our certificate with a passphrase. We need Apache to be able to read the file, without user intervention, when the server starts up. A passphrase would prevent this from happening because we would have to enter it after every restart.

-days 365: This option sets the length of time that the certificate will be considered valid. We set it for one year here.

-newkey rsa:2048: This specifies that we want to generate a new certificate and a new key at the same time. We did not create the key that is required to sign the certificate in a previous step, so we need to create it along with the certificate. The `rsa:2048` portion tells it to make an RSA key that is 2048 bits long.

-keyout: This line tells OpenSSL where to place the generated private key file that we are creating.

-out: This tells OpenSSL where to place the certificate that we are creating. 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.

Fill out the prompts appropriately. The most important line is the one that requests the Common Name (e.g. server FQDN or YOUR name). You need to enter the domain name associated with your server or, more likely, your server's public IP address.

The entirety of the prompts will look something like this:

Output

```
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:New York
Locality Name (eg, city) []:New York City
```

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Bouncy Castles, Inc.
Organizational Unit Name (eg, section) []:Ministry of Water Slides
Common Name (e.g. server FQDN or YOUR name) []:server_IP_address
Email Address []:admin@your_domain.com
Both of the files you created will be placed in the appropriate subdirectories under /etc/ssl.

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.

We will make a few adjustments to our configuration:

We will create a configuration snippet to specify strong default SSL settings. We will modify the included SSL Apache Virtual Host file to point to our generated SSL certificates.

(Recommended) We will modify the unencrypted Virtual Host file to automatically redirect requests to the encrypted Virtual Host.

When we are finished, we should have a secure SSL configuration.

Creating an Apache Configuration Snippet with Strong Encryption Settings

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
```

To set up Apache SSL securely, we will be using the recommendations by Remy van Elst on the Cipherli.st site. This site is designed to provide easy-to-consume encryption settings for popular software.

The suggested settings on the site linked to above offer strong security. Sometimes, this comes at the cost of greater client compatibility. If you need to support older clients, there is an alternative list that can be accessed by clicking the link on the page labelled “Yes, give me a ciphersuite that works with legacy / old software.” That list can be substituted for the items copied below.

The choice of which config you use will depend largely on what you need to support. They both will provide great security.

For our purposes, we can copy the provided settings in their entirety. We will just make one small change to this and disable the Strict-Transport-Security header (HSTS).

Preloading HSTS provides increased security, but can have far-reaching consequences if accidentally enabled or enabled incorrectly. In this guide, we will not enable the settings, but you can modify that if you are sure you understand the implications.

Before deciding, take a moment to read up on HTTP Strict Transport Security, or HSTS, and specifically about the “preload” functionality.

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

```
/etc/apache2/conf-available/ssl-params.conf
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
```

Inside, with most of the comments removed, the Virtual Host block should look something like this by default:

```
/etc/apache2/sites-available/default-ssl.conf
<IfModule mod_ssl.c>
    <VirtualHost _default_:443>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www/html

        ErrorLog ${APACHE_LOG_DIR}/error.log
        CustomLog ${APACHE_LOG_DIR}/access.log combined

        SSLEngine on

        SSLCertificateFile      /etc/ssl/certs/ssl-cert-snakeoil.pem
        SSLCertificateKeyFile   /etc/ssl/private/ssl-cert-snakeoil.key

        <FilesMatch "\.(cgi|shtml|phtml|php)$">
            SSLOptions +StdEnvVars
        </FilesMatch>
        <Directory /usr/lib/cgi-bin>
            SSLOptions +StdEnvVars
        </Directory>

    </VirtualHost>
</IfModule>
```

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. Again, if you're using a different document root, be sure to update the DocumentRoot directive.

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.

(Recommended) Modifying the HTTP Host File to Redirect to HTTPS

As it stands now, the server will provide both unencrypted HTTP and encrypted HTTPS traffic. For better security, it is recommended in most cases to redirect HTTP to HTTPS automatically. If you do not want or need this functionality, you can safely skip this section.

To adjust the unencrypted Virtual Host file to redirect all traffic to be SSL encrypted, 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, 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>

```

Save and close the file when you are finished.

That's all of the configuration changes you need to make to Apache. Next, we will discuss how to update firewall rules with ufw to allow encrypted HTTPS traffic to your server.

Step 3 – Adjusting the Firewall

If you have the ufw firewall enabled, as recommended by the prerequisite guides, you might need to adjust the settings to allow for SSL traffic. Fortunately, when installed on Debian 10, ufw comes loaded with app profiles which you can use to tweak your firewall settings

We can see the available profiles by typing:

```
sudo ufw app list
```

You should see a list like this, with the following four profiles near the bottom of the output:

Output

Available applications:

```
. . .
WWW
WWW Cache
WWW Full
WWW Secure
```

. . .
You can see the current setting by typing:

```
sudo ufw status
```

If you allowed only regular HTTP traffic earlier, your output might look like this:

Output

Status: active

To	Action	From
--	-----	----
OpenSSH	ALLOW	Anywhere
WWW	ALLOW	Anywhere
OpenSSH (v6)	ALLOW	Anywhere (v6)
WWW (v6)	ALLOW	Anywhere (v6)

To additionally let in HTTPS traffic, allow the "WWW Full" profile and then delete the redundant "WWW" profile allowance:

```
AIM
Bonjour
CIFS
DNS
Deluge
Dovecot IMAP
Dovecot POP3
Dovecot Secure IMAP
Dovecot Secure POP3
IMAP
IMAPS
IPP
KTorrent
Kerberos Admin
Kerberos Full
Kerberos KDC
Kerberos Password
LDAP
LDAPS
LPD
MSN
MSN SSL
Mail submission
NFS
OpenSSH
POP3
POP3S
PeopleNearby
Postfix
Postfix SMTPS
Postfix Submission
SMTP
SSH
Socks
Telnet
Transmission
Transparent Proxy
VNC
WWW
WWW Cache
```



```
WWW Full
WWW Secure
XMPP
Yahoo
qBittorrent
svnserve
```

```
sudo ufw allow 'WWW Full'
sudo ufw delete allow 'WWW'
Your status should look like this now:
```

```
sudo ufw status
Output
Status: active
```

To	Action	From
--	-----	----
OpenSSH	ALLOW	Anywhere
WWW Full	ALLOW	Anywhere
OpenSSH (v6)	ALLOW	Anywhere (v6)
WWW Full (v6)	ALLOW	Anywhere (v6)

With your firewall configured to allow HTTPS traffic, you can move on to the next step where we'll go over how to enable a few modules and configuration files to allow SSL to function properly.

Step 4 – Enabling the Changes in Apache

Now that we've made our changes and adjusted our firewall, we can enable the SSL and headers modules in Apache, enable our SSL-ready Virtual Host, and then restart Apache to put these changes into effect.

Enable `mod_ssl`, the Apache SSL module, and `mod_headers`, which is needed by some of the settings in our SSL snippet, with the `a2enmod` command:

```
sudo a2enmod ssl
sudo a2enmod headers
```

Next, enable your SSL Virtual Host with the `a2ensite` command:

```
sudo a2ensite default-ssl
```

You will also need to enable your `ssl-params.conf` file, to read in the values you've set:

```
sudo a2enconf ssl-params
```

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

```
sudo apache2ctl configtest
```

If everything is successful, you will get a result that looks like this:

Output

```
Syntax OK
```

As long as your output has Syntax OK in it, then your configuration file has no syntax errors and you can safely restart Apache to implement the changes:

```
sudo systemctl restart apache2
```

With that, your self-signed SSL certificate is all set. You can now test that your server is correctly encrypting its traffic.

Step 5 – Testing Encryption

You're now ready to test your SSL server.

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 you 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:

Apache self-signed cert warning

This is expected and normal. We are only interested in the encryption aspect of our certificate, not the third party validation of our host's authenticity. Click ADVANCED and then the link provided to proceed to your host anyways:

Apache self-signed override

You should be taken to your site. If you look in the browser address bar, you will see a lock with an "x" over it or another similar "not secure" notice. In this case, this just means that the certificate cannot be validated. It is still encrypting your connection.

If you configured Apache to redirect HTTP to HTTPS, you can also check whether the redirect functions correctly:

`http://server_domain_or_IP`

If this results in the same icon, this means that your redirect worked correctly. However, the redirect you created earlier is only a temporary redirect. If you'd like to make the redirection to HTTPS permanent, continue on to the final step.

Step 6 – Changing to a Permanent Redirect

If your redirect worked correctly and you are sure you want to allow only encrypted traffic, you should modify the unencrypted Apache Virtual Host again to make the redirect permanent.

Open your server block configuration file again:

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

Find the Redirect line we added earlier. Add permanent to that line, which changes the redirect from a 302 temporary redirect to a 301 permanent redirect:

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

```
<VirtualHost *:80>
```

```
    . . .
```

```
        Redirect permanent "/" "https://your_domain_or_IP/"
```

```
    . . .
```

```
</VirtualHost>
```

Save and close the file.

Check your configuration for syntax errors:

```
sudo apache2ctl configtest
```

If this command doesn't report any syntax errors, restart Apache:

```
sudo systemctl restart apache2
```

This will make the redirect permanent, and your site will only serve traffic over HTTPS.

Conclusion

You have configured your Apache server to use strong encryption for client connections. This will allow you serve requests securely, and will prevent outside parties from reading your traffic.