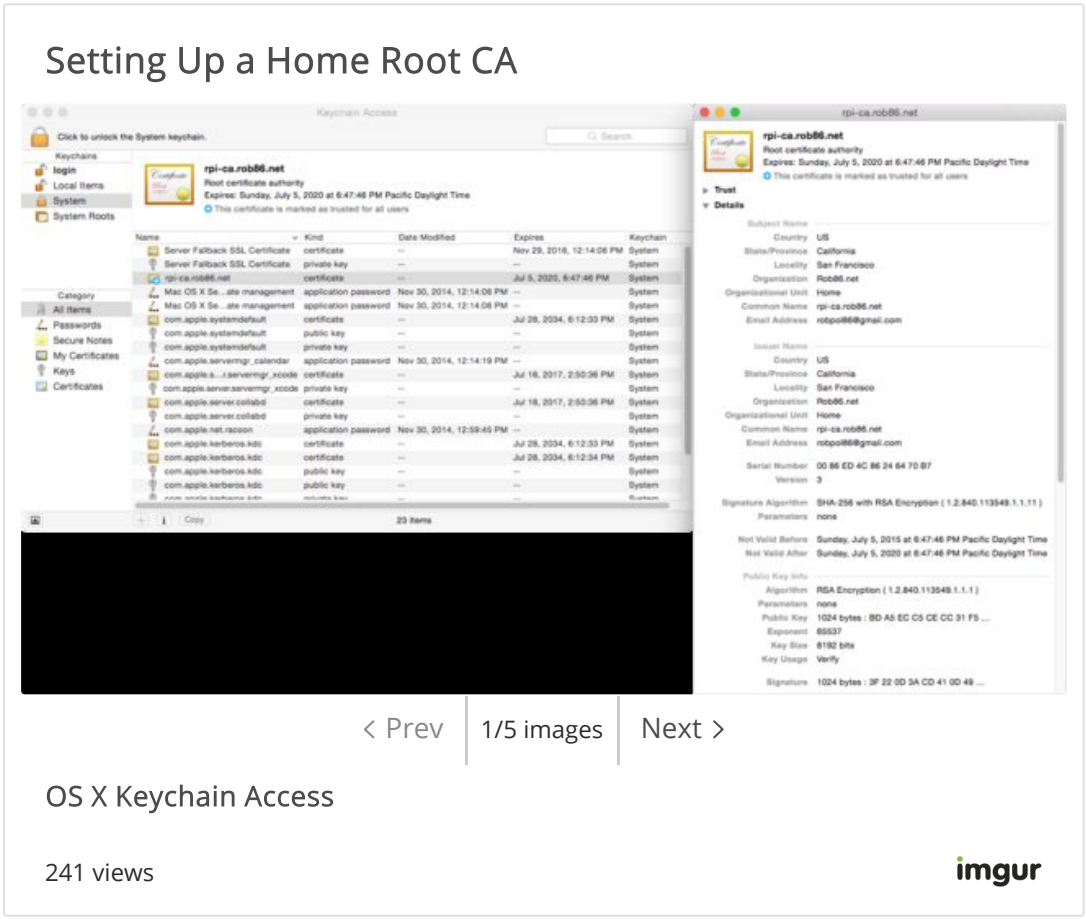
Setting Up a Home Root CA

Tired of getting those SSL error pages when accessing your router's admin interface? Sick of having to click three times to get to your IPMI web interface? Have I got a guide for you!



This guide will go over setting up an offline root certificate authority for your home network. It is based on what I've learned from https://jamielinux.com/docs/openssl-certificate-authority/index.html with a few differences:

1. We will not be creating an intermediate pair here. Since my intentions are just setting up SSL certs on a handful of internal web interfaces and maybe even WPA2 Enterprise one day, I didn't think it was worth setting this up. It might make revoking certs not as quick, but I don't see myself signing very many certs after my initial run.
2. I'll include steps on how to bridge the air gap. For maximum paranoid-tier security we will not be plugging in any USB flash drives (or USB anything excluding keyboards) or network cables. WiFi adapters are also obviously forbidden. For this we'll be using qrencode.
3. I'll be assuming the Linux computer you're using has a GUI (desktop environment). This is to reduce the number of QR codes needed since you'll have more resolution with a GUI than with a frame buffer so you can fit more data in each QR code.
4. For additional paranoid-tier security we'll generate a 8192-bit long RSA key for our root CA. 4096-bit keys are fine too but I'm crazy. We'll also be creating 4096-bit SSL keys instead of the usual 2048-bit. If you're using OS X and you get an error trying to install the root certificate, read: https://apple.stackexchange.com/questions/110261/mac-os-x-10-9-and-8192-bit-certificates-error-67762/

While this guide should work fine with any Linux computer I'll be focusing on Debian-based distributions. This guide has been tested on **Debian Jessie** on an old T60 Thinkpad and **2017-01-11-raspbian-jessie.zip** on a Raspberry Pi.

The goal here is to setup an offline root CA. It will be online at first to get updates but right before generating the root pair we will remove any network connectivity from the host and never EVER connect it to any networks or USB devices. This will be an offline and air gapped root CA.

Preparing the Host

This section will go over preparing a newly-installed Debian/Raspbian system. For machines without a real time clock (e.g. Raspberry Pis) we'll setup a script that runs during boot that prompts you for the current time.

1. Perform a clean install of Debian (or install the latest Raspbian PIXEL image on the Raspberry Pi) and boot up the host. It's ok to have network access for now. For my Raspberry Pi I followed Raspbian Setup (Raspberry Pi) (you don't need to install any of those packages in that link, just upgrade).
   A. If you're using Debian be sure to create an **encrypted** LVM or partition.
   B. On a Raspberry Pi you can follow my guide to encrypt the main partition: Raspberry Pi LUKS Root Encryption

2. Upgrade all of your packages since this will be the last time the system will have internet access: `sudo apt-get update && sudo apt-get upgrade`.
3. `sudo reboot` in case a new kernel was installed.
4. Finally install these required packages: `sudo apt-get install qrencode acl`

Boot Date Prompt on Raspberry Pi

Raspberry Pis don't have real time clocks so they don't keep track of the time when powered off. Usually they handle this by getting the current time from the internet after booting up. However since our root CA will never have internet access again we need to always set the current time every time it boots up.

Since time is very important for signing certificates we'll want to avoid forgetting this. You can install this systemd file to have it prompt you for the current time before the Raspberry Pi finishes booting up, guaranteeing you won't forget:

```
# Prompt for current date during boot.
#
# https://github.com/Robpol86/robpol86.com/blob/master/docs/_static/date-prompt.service
#
# Stops the system from booting up to 2 minutes or until user enters the
# current time in the console. Useful for Raspberry Pis and other systems
# with no real time clocks.
#
# Save as: /etc/systemd/system/date-prompt.service
# Enable with: systemctl enable date-prompt.service

[Unit]
After=fake-hwclock.service
After=systemd-fsck-root.service systemd-fsck@dev-mmcblk0p1.service
Before=sysinit.target plymouth-start.service
DefaultDependencies=no
Description=Prompt for current date during boot

[Service]
Environment="P=Enter the current date (e.g. Feb 11 5:17 PM): "
ExecStartPre=-/bin/plymouth deactivate
ExecStart=/bin/bash -c 'until read -ep "$P" R; [ ! -z "$R" ] && date -s "$R"; do :; done'
ExecStartPost=-/bin/plymouth reactivate
ExecStopPost=-/bin/plymouth reactivate
RemainAfterExit=yes
StandardError=inherit
StandardInput=tty
StandardOutput=inherit
TimeoutSec=120
Type=oneshot

[Install]
WantedBy=sysinit.target
```

Copy the OpenSSL Config

Note

> Based on a few articles I've found while considering which domain to use at home, I thought I would mention it here even though it's more of a network-related topic rather than an SSL/Certificate topic. I highly encourage you to either purchase a dedicated domain name for your home network or at least use a dedicated subdomain on a domain you already own.
>
> In the table below I'll use `myhome.net` as an example. Org Name is just a name so in this case the value would be "MyHome.net". If you used `home.mycooldomain.com` then the Org Name equivalent may be "Home.MyCoolDomain.com". It can actually be set to anything but this is what I've done for my home network.

The first step is to configure OpenSSL. You'll need to replace some values in the configuration file I'll be providing to you. Refer to the table below for what you'll be replacing.

To Replace	Replace With	Example
SUB_COUNTRY_NAME	Two-letter ISO abbreviation for your country.	US
SUB_STATE_NAME	State or province where you live. No abbreviations.	California
SUB_LOCALITY	City where you are located.	San Francisco
SUB_ORG_NAME	Name of your organization.	MyHome.net
SUB_UNIT_NAME	Section of the organization.	Home
SUB_EMAIL	Your contact email.	xx@yy.zz

Overwrite all of `/etc/ssl/openssl.cnf` with the following (it's still ok to have network access for this part). Be sure to replace `SUB_` strings.

```
# /etc/ssl/openssl.cnf

CN = ""  # Leave blank.

[ ca ]
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir               = /root/ca
certs             = $dir/certs
crl_dir           = $dir/crl
new_certs_dir     = $dir/newcerts
database          = $dir/index.txt
serial            = $dir/serial
RANDFILE          = $dir/private/.rand

# The root key and root certificate.
private_key       = $dir/private/ca.key.pem
certificate       = $dir/certs/ca.cert.pem

# For certificate revocation lists.
crlnumber         = $dir/crlnumber
crl               = $dir/crl/ca.crl.pem
crl_extensions    = crl_ext
default_crl_days  = 30

default_md        = sha256
name_opt          = ca_default
cert_opt          = ca_default
default_days      = 375
preserve          = no
policy            = policy_loose

[ policy_loose ]
# See the POLICY FORMAT section of the `ca` man page.
countryName             = optional
stateOrProvinceName     = optional
localityName            = optional
organizationName        = optional
organizationalUnitName  = optional
commonName              = supplied
```

```
emailAddress            = optional

[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 4096
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

# Optionally, specify some defaults.
countryName_default             = SUB_COUNTRY_NAME
stateOrProvinceName_default     = SUB_STATE_NAME
localityName_default            = SUB_LOCALITY
0.organizationName_default      = SUB_ORG_NAME
organizationalUnitName_default  = SUB_UNIT_NAME
commonName_default              = $ENV::CN
emailAddress_default            = SUB_EMAIL

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectAltName = DNS:$ENV::CN
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true, pathlen:0
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = "OpenSSL Generated Client Certificate"
```

```
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, emailProtection

[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectAltName = DNS:$ENV::CN
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[ crl_ext ]
# Extension for CRLs (`man x509v3_config`).
authorityKeyIdentifier=keyid:always

[ ocsp ]
# Extension for OCSP signing certificates (`man ocsp`).
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, digitalSignature
extendedKeyUsage = critical, OCSPSigning
```

OpenSSL Directory Structure

Everything will live in `/root/ca`. It will also all be owned by root. Remember this computer is a dedicated CA so it won't be doing anything else at all except hosting your very important root certificate private key and the root certificate itself.

Run these commands to setup directories and permissions:

```
sudo mkdir -p /root/ca/{certs,crl,csr,newcerts,private}
sudo setfacl -d -m u::rx -m g::- -m o::- /root/ca/private
sudo setfacl -d -m u::rx -m g::rx -m o::rx /root/ca/certs
sudo chmod 700 /root/ca/private
sudo touch /root/ca/index.txt
sudo tee /root/ca/serial <<< 1000
```

Those `setfacl` commands set filesystem ACLs which enforce default maximum file permissions for new files/directories. A brief description for these directories:

Directory	Description
`/root/ca/certs`	Certificates are dumped here.
`/root/ca/crl`	Certificate revocation lists.
`/root/ca/csr`	Certificate signing request.
`/root/ca/newcerts`	Not used in this guide.
`/root/ca/private`	Private keys. VERY SENSITIVE.

Air Gap

This is the moment we've all been waiting for! Just copy one more file and then isolate the host from the world permanently.

Place the following file into `/usr/local/bin/airgap` and `chmod +x` it. We'll use this script to convert files into QR codes after compressing and encrypting them.

```bash
#!/bin/bash

# Tar up and encrypt specified files before encoding them to QR codes.
#
# https://github.com/Robpol86/robpol86.com/blob/master/docs/_static/airgap.sh
#
# Using large QR codes to hop the air gap and transmit files out of this host
# with no network connectivity.
# Generate and print a temporary password used to encrypt the file contents.
# Save as (chmod +x): /usr/local/bin/airgap
#
# Reconstruct data on OS X with the following:
# 1. Read QR codes from this host's screen using phone app.
# 2. Have phone save text data to Dropbox or local storage and transfer to PC.
# 3. Save files as 1.txt, 2.txt, etc.
# 4. cat [1-3].txt |base64 -D |openssl enc -aes-256-cfb -d |tar -xzv

set -e  # Exit script if a command fails.
set -u  # Treat unset variables as errors and exit immediately.
set -o pipefail  # Exit script if pipes fail instead of just the last program.

# Handle no arguments specified.
if [[ $# -eq 0 ]]; then
    echo "Must specify relative path to files to encode."
    exit 1
fi

# Handle bad arguments.
declare -A visited
for file in "$@"; do
    if [ ${visited[$file]+true} ]; then
        echo "ERROR: file $file specified more than once."
        exit 2
    elif [ ! -e "$file" ]; then
        echo "ERROR: file $file does not exist."
        exit 2
    elif [ ! -f "$file" ]; then
        echo "ERROR: file $file is not a file."
        exit 2
    elif [ ! -r "$file" ]; then
        echo "ERROR: file $file is not readable."
        exit 2
```

```
    fi
    visited[$file]=1
done

# Generate random password 10 to 15 characters in length.
export PASSWORD=$(openssl rand -base64 25 |cut -c -$((10+RANDOM%6)))

# Remove any previously created QR codes.
rm -vf /tmp/qr*.png

# Encode.
echo -e "\x1b[1mCompressing, encrypting, and encoding $# file(s)...\x1b[0m"
tar -czv $@ |
    openssl enc -aes-256-cfb -salt -pass env:PASSWORD |
    base64 -w0 |
    qrencode -o /tmp/qr.png -Sv40
echo -e "\x1b[1mDone\x1b[0m"

# Print.
ls -l /tmp/qr*.png
echo -en "\x1b[1mPassword:\x1b[0m "
echo "$PASSWORD"
```

Now remove all USB devices (sans keyboard) and network cables/connections. If this is on a Raspberry Pi either swap it out with a Model A (the one without WiFi or ethernet ports), or fill in the ethernet port with hot glue. Do the same with all but one USB ports. Or just be super duper sure never to plug in things when using this SD card.

Finally Generate the Pair

This is where we actually generate the root key and certificate. The root key is used to sign additional certificate pairs for specific devices/servers, and the root certificate is what you'll export to clients that should trust any of these additional certificates.

Warning

The root key `ca.key.pem` you'll be generating is the most sensitive file on this dedicated computer. Keep it as secure as possible. When `openssl genrsa` asks you for a password enter a unique and very secure password. Make sure `setfacl` worked and the permissions are:

```
-r-------- 1 root root 1.8K Aug 15 12:21 private/ca.key.pem
```

Note

The `openssl req` command will prompt you for some information. The defaults you've specified in openssl.cnf will be fine. However double check that the **Common Name** is the fully qualified domain name of this certificate authority.

```
sudo su -  # Become root.
cd /root/ca
export CN=$(hostname --fqdn)
openssl genrsa -aes256 -out private/ca.key.pem 8192
openssl req -key private/ca.key.pem -new -x509 -days 1827 -extensions v3_ca -out certs/ca.cert.pem
openssl x509 -noout -text -in certs/ca.cert.pem |more  # Confirm everything looks good.
```

You're done generating your root certificate and private key. You're technically "done". However you'll probably want to do these two steps:

1. Install the public root certificate on client computers so they can trust your servers instead of getting SSL errors.
2. Creating an SSL certificate to install on your web servers (router admin pages, IPMI interfaces, etc.). For more info see: Issuing Server Certificates

For the former you'll want to export the `certs/ca.cert.pem` file and install it on client computers/devices. For example:

- OS X: The **Keychain Access** app can install that file in the System keychain (not System Roots), an you'll need to manually set the trust to "Always Trust" (ou may also have to restart web browsers or just reboot to get rid of SSL errors).
- Fedora/CentOS/RHEL: Copy that file to `/etc/pki/ca-trust/source/anchors/` and then run `sudo update-ca-trust`.

Instructions for exporting this file is available in the Bridging the Air Gap section below.

Frequent Tasks

This section will contain additional sub sections with instructions on how to complete some tasks you may repeat for different use cases.

Bridging the Air Gap

We can use the `airgap` script we copied earlier to encode files into one or more QR codes to be scanned by your phone and reassembled on another computer. This is a one-way data transfer so your root CA host remains secure and air gapped.

> **Note**
>
> `airgap` will print a password at the end of its run. Use this one-time password to decrypt the files on the receiving computer.

For example if you want to just export the `certs/ca.cert.pem` file you'll do something like this (you can also specify multiple files for airgap to encode at once):

```
pi@raspberrypi:~ $ sudo su -
root@raspberrypi:~# cd /root/ca
root@raspberrypi:~/ca# airgap certs/ca.cert.pem
Compressing, encrypting, and encoding 1 file(s)...
certs/ca.cert.pem
Done
-rw-r--r-- 1 root root 7219 Feb 16 16:28 /tmp/qr-01.png
-rw-r--r-- 1 root root 6816 Feb 16 16:28 /tmp/qr-02.png
Password: 3SOD8voj8bT
root@raspberrypi:~/ca#
```

Then in the GUI open both of those files and scan them with your phone using a QR scanner app. If you're scanning QR codes with an Android phone using Barcode Scanner you can "Share via email" which gives you the option to share to Dropbox (for some dumb reason) which makes it easy to get encrypted data on your computer.

Reconstructing Data

Once you've scanned the QR codes and saved the large strings of data somewhere on a Linux or OS X computer run these commands to reassemble and decrypt data:

```
mkdir ~/inbound_certs
cat [1-3].txt |base64 -D |openssl enc -aes-256-cfb -d |tar -xzvC ~/inbound_certs
```

Issuing Server Certificates

This section covers issuing SSL certificates for web servers such as router admin pages. We will generate an SSL certificate and its private key. You'll need to install both files on the web server. Keep in mind the private key is very sensitive and is used to sign SSL sessions to keep it secure as you transfer it to the web server!

Note

When asked for a **Common Name** you'll need to enter the web server's FQDN. So instead of accessing your router admin page using http://192.168.0.1 you'll instead be using https://router.myhome.net for example. Common Name here will be `router.myhome.net`.

On the root CA host run these commands. Substitute `router.myhome.net` with whatever FQDN your target web server will use.

```
date  # Verify the date is correct. If not: sudo date -s "Aug 15 18:10"
sudo su -
cd /root/ca
export CN=router.myhome.net
openssl genrsa -out private/$CN.key.pem 4096
openssl req -key private/$CN.key.pem -new -out csr/$CN.csr.pem  # CN is FQDN.
openssl ca -extensions server_cert -notext -in csr/$CN.csr.pem -out certs/$CN.cert.pem
rm csr/$CN.csr.pem
openssl x509 -noout -text -in certs/$CN.cert.pem |more  # Confirm everything looks good.
cat index.txt  # Verify new cert is present.
```

Verify that the **Issuer** is the root CA and the **Subject** is the certificate itself. You will need to install both `certs/router.myhome.net.cert.pem` and `private/router.myhome.net.key.pem` on the web server. Read Bridging the Air Gap for instructions on how to do this securely.

Adding to Java Certificate Store

Java seems to have its own certificate store separate from the Operating System's. So in the case of IPMI you'll have a valid HTTPS connection to the web interface but when trying to open the Console Redirection you'll get certificate warnings.

To fix this you'll need to add the host's (not root) certificate to Java's certificate store. On OS X:

1. Open **System Preferences**. Click on the **Java** icon at the bottom.
2. In the new window click on the **Security** tab and then the **Manage Certificates** button at the bottom.
3. Set certificate type to **Secure Site CA**.
4. Click Import, set File Format to **All Files**, and import `router.myhome.net.cert.pem`.
5. Click Close and OK.

Comments