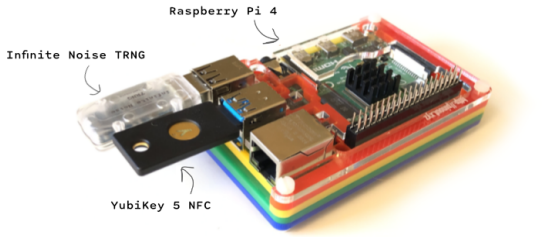
smallstep.com

Build a Tiny Certificate Authority For Your Homelab

Why would I want a Certificate Authority in my homelab?!

22-28 minutes



TL;DR In this tutorial, we're going to build a tiny, standalone, online Certificate Authority (CA) that will mint TLS certificates and is secured with a YubiKey. It will be an internal ACME server on our local network (ACME is the same protocol used by Let's Encrypt). The YubiKey will securely store the CA private keys and sign certificates, acting as a cheap alternative to a Hardware Security Module (HSM). We'll also use an open-source True Random Number Generator, called Infinite Noise TRNG, to spice up the Linux entropy pool.

- Because end-to-end TLS is great and you should easily be able to run TLS wherever you need it. Especially in your homelab. Internal networks are no longer perceived as a safe zone where unencrypted traffic is okay. But you need certificates.
- Because TLS client authentication is becoming more widely supported in different services, and it's a lot better than passwords. But you need certificates.
- Because the ACME protocol (used by Let's Encrypt) can easily be deployed internally, so you can automate renewal and never have to think about your certificates.
- Because maybe you've done the 'self-signed certificate' rigmarole with OpenSSL a dozen times already. Might as well formalize things and get your devices to trust a CA that you can use wherever you need it.
- Because setting up a simple CA is a great learning experience.

Tiny CA Specs

- Raspberry Pi 4 Model B 2GB + microSD card
- Any YubiKey that supports the Personal Identity Verification (PIV) application, for CA signing operations. I'm using a YubiKey 5 NFC.
- Optional: Infinite Noise TRNG for outboard random number generation.
- A USB thumb drive—or a second YubiKey—for storing an offline backup of our CA
- We'll be running the `step-ca` open-source online Certificate Authority.
- Total cost: Around US$100

Part 1: System Setup

Basic OS & Networking Setup

- On your laptop, burn the Ubuntu 20.10 Server 64-bit ARM pre-installed server image onto the microSD card using the Raspberry Pi Imager.
- Fire up the Raspberry Pi, plug it into your network, and find its initial IP address. You can run `arp -na | grep -e "b8:27:eb" -e "dc:a6:32" -e "e4:5f:01"` to discover Raspberry Pi devices on the local network.
- Login via SSH (username and password will be `ubuntu`), and change the password.
- Set the hostname via `hostnamectl set-hostname tinyca`
- Set the timezone using `timedatectl set-timezone America/Los_Angeles` (or whatever your timezone is; `timedatectl list-timezones` will list them all)
- Be sure NTP is working. Check status with `timedatectl`— make sure "NTP Service" is "active". If not, you can add some NTP servers to `/etc/systemd/timesyncd.conf` and run `systemctl restart systemd-timesyncd`.
- You'll need the machine to have a DNS name (for me it's `tinyca.internal`) and/or a static IP on your network.

  Now that you have good time synchronization and a stable hostname, we can proceed.

  Install prerequisite: `ykman`

  Now, insert your YubiKey. Let's install the `yubikey-manager` (and dependency `pcscd`) and make sure you can connect to the YubiKey:

```
$ sudo apt update
$ sudo apt install -y yubikey-manager
$ ykman info
Device type: YubiKey 5 NFC
Serial number: 13910388
Firmware version: 5.2.7
Form factor: Keychain (USB-A)
Enabled USB interfaces: OTP+FIDO+CCID
NFC interface is enabled.
```

  Install prerequisite: Go

  You'll need Go in order to build the `step-ca` server.

```
$ cd
$ curl -LO https://golang.org/dl/go1.15.6.linux-arm64.tar.gz
$ sudo tar -C /usr/local -xzf go1.15.6.linux-arm64.tar.gz
$ echo "export PATH=\$PATH:/usr/local/go/bin" >> .profile
$ source .profile
$ go version
go version go1.15.6 linux/arm64
```

  Build and install `step-ca` and `step`

  You'll need to install both `step-ca` (the CA server software) and `step` (the command used to configure and control `step-ca`).

  First, download the source for `step-ca` and build it with experimental YubiKey support enabled:

```
$ curl -LO https://github.com/smallstep/certificates/archive/v0.15.5.tar.gz
$ tar xvzf v0.15.5.tar.gz
```

```
$ cd certificates-0.15.5/

# step-ca instructions start here:
$ sudo apt-get install -y libpcsclite-dev gcc make
pkg-config
$ make bootstrap
$ make build GOFLAGS=""
....
Build Complete!
$ sudo cp bin/step-ca /usr/local/bin
$ sudo setcap CAP_NET_BIND_SERVICE=+eip /usr/local
/bin/step-ca
$ step-ca version
Smallstep CA/0.15.5 (linux/arm64)
Release Date: 2020-12-08 19:49 UTC
```

Now install `step` from a prebuilt binary:

```
$ curl -LO https://github.com/smallstep
/cli/releases/download/v0.15.3
/step_linux_0.15.3_arm64.tar.gz
$ tar xvzf step_linux_0.15.3_arm64.tar.gz
$ sudo cp step_0.15.3/bin/step /usr/local/bin
$ step version
Smallstep CLI/0.15.3 (linux/arm64)
Release Date: 2020-10-21 23:46 UTC
```

Optional, but 🔥 Set up the outboard random number generator

Infinite Noise TRNG is an open-source USB True Random Number Generator. It uses a "modular entropy multiplier" architecture to generate a *lot* of random data quickly. For this setup, a daemon will continuously feed entropy into Linux's system entropy pool by writing to `/dev/random`.

> **But will this lovely new entropy generator actually be used by the CA?** I needed to answer two questions here:
>
> 1. How does the CA generate random numbers? I had to dig around a little to confirm this. `step-ca` uses Go's `crypto/rand` for all of its key generation, and `cyrpto/rand` uses `/dev/urandom` as its random data source on Linux systems.
> 2. Does the entropy created via writing to `/dev/random` actually affects what is read from `/dev/urandom`? It does—because Linux has only one entropy pool, shared by `/dev/random` and `/dev/urandom`.
>
> We also need to confirm that the outboard TRNG is actually generating high quality noise. We'll do that in a minute.

You'll need to compile the driver from source, because there's no pre-built `arm64` package available.

```
$ curl -LO https://github.com/13-37-org/infnoise
/archive/0.3.1.tar.gz
$ tar xvzf 0.3.1.tar.gz
$ cd infnoise-0.3.1/software
$ sudo apt-get install -y libftdi-dev libusb-dev
$ make -f Makefile.linux
$ sudo make -f Makefile.linux install
install -d /usr/local/sbin
install -m 0755 infnoise /usr/local/sbin/
install -d /usr/local/lib/udev/rules.d/
install -m 0644 init_scripts/75-infnoise.rules
/usr/local/lib/udev/rules.d/
install -d /usr/local/lib/systemd/system
```

```
install -m 0644 init_scripts/infnoise.service
/usr/local/lib/systemd/system
```

Now, plug in the TRNG and restart your system.

```
$ sudo reboot
```

After a restart, you should see that the driver has started up. It will start and stop based on whether the TRNG is present.

```
$ systemctl status infnoise
● infnoise.service - Wayward Geek InfNoise TRNG
driver
     Loaded: loaded (/usr/local/lib/systemd/system
/infnoise.service; disabled; vendor preset:
enabled)
     Active: active (running) since Tue 2020-12-08
12:52:51 PST; 2min 0s ago
    Process: 1652 ExecStart=/usr/local
/sbin/infnoise --dev-random --daemon --pidfile
/var/run/infnoise.pid (code=exited, status=0>
   Main PID: 1657 (infnoise)
      Tasks: 1 (limit: 2099)
     CGroup: /system.slice/infnoise.service
             └─1657 /usr/local/sbin/infnoise
--dev-random --daemon --pidfile /var/run
/infnoise.pid

Dec 08 12:52:51 tinyca systemd[1]: Starting
Wayward Geek InfNoise TRNG driver...
Dec 08 12:52:51 tinyca systemd[1]: Started Wayward
Geek InfNoise TRNG driver.
```

Finally, let's run a health check to make sure the TRNG is ready for use:

```
$ infnoise --debug --no-output
Generated 1048576 bits.  **OK to use data.**
Estimated entropy per bit: 0.877159, estimated K:
1.836755
num1s:50.514668%, even misfires:0.111168%, odd
misfires:0.152342%
^C
```

Entropy is written to `/dev/random` by `infnoise.service` every second. You're all set on randomness! Now that you have more than enough entropy, you're ready to generate your CA keys.

Part 2: Creating Your PKI

Now you'll create your root and intermediate CA certificates and keys, and store them securely on the YubiKey.

Ideally, your Raspberry Pi should be kept offline for this section. Disconnect the Ethernet cable, and connect directly to the device via HDMI and a keyboard.

Prepare a USB thumb drive for storing the private keys

You can't just have your CA private keys live *only* on the YubiKey. You'll want at least one backup of them, in case the YubiKey breaks!

Insert a USB thumb drive. You'll generate the keys directly on this drive, so that they never touch the Pi's microSD card. First, find the device name of your USB drive:

```
$ sudo fdisk -l
...
Disk /dev/sda: 14.91 GiB, 16005464064 bytes,
```

```
31260672 sectors
Disk model: Cruzer Fit
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512
bytes
...
```

In this case, the drive is `/dev/sda`. Let's initialize it with a single `ext4` partition:

```
$ sudo fdisk /dev/sda
Welcome to fdisk (util-linux 2.36).
Changes will remain in memory only, until you
decide to write them.
Be careful before using the write command.

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-31260671, default 2048):
Last sector, +/-sectors or +/-size{K,M,G,T,P}
(2048-31260671, default 31260671):

Created a new partition 1 of type 'Linux' and of
size 14.9 GiB.

Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.

$ sudo mkfs.ext4 /dev/sda1 -v
mke2fs 1.45.6 (20-Mar-2020)
fs_types for mke2fs.conf resolution: 'ext4'
Filesystem label=
OS type: Linux
...
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting
information: done
```

Generate your PKI on the thumb drive

Great, now you're ready to create your Public Key Infrastructure (PKI). Specifically, you'll be creating CA keys and certificates.

Tiny CA PKI Highlights:

- Tiny CA has a root CA key and certificate, and an intermediate CA key and certificate.
- The root CA key signs the Intermediate CA certificate.
- The root CA certificate is self-signed (signed with the root CA key)
- The intermediate CA key will sign all of your TLS certificates.
- By default, `step-ca` issues certificates with a 24-hour lifetime. I hope this default will compel you to set up automated renewal on your clients. And you can always increase the TLS certificate duration in the CA configuration, if you want something a bit more relaxed.
- If a device is configured to trust your root CA, it will trust certificates you create with `step-ca`.

- You can throw away the root CA key if you never need another intermediate.
- Need a refresher on X.509 certificates? See our post, Everything you should know about certificates and PKI but are too afraid to ask.

Use a strong password when prompted, and save your password separately, offline, somewhere super duper safe.

```
$ sudo mount /dev/sda1 /mnt
$ cd /mnt
$ sudo mkdir ca
$ sudo chown ubuntu:ubuntu ca
$ export STEPPATH=/mnt/ca
$ step ca init --pki --name="Tiny"
✔ What do you want your password to be? [leave
empty and we'll generate one]: ...
Generating root certificate...
all done!

Generating intermediate certificate...
all done!

✔ Root certificate: /mnt/ca/certs/root_ca.crt
✔ Root private key: /mnt/ca/secrets/root_ca_key
✔ Root fingerprint:
d6b3b9ef79a42aeeabcd5580b2b516458ddb25d1af4ea7ff0845e624ec1bb609
✔ Intermediate certificate: /mnt/ca/certs
/intermediate_ca.crt
✔ Intermediate private key: /mnt/ca/secrets
/intermediate_ca_key

FEEDBACK 😍 🍻
      The step utility is not instrumented for
usage statistics. It does not
      phone home. But your feedback is extremely
valuable. Any information you
      can provide regarding how you're using
`step` helps. Please send us a
      sentence or two, good or bad:
feedback@smallstep.com or join
      https://github.com/smallstep/certificates
/discussions.
```

Don't forget to give your CA a cute name! It will appear on all of your certificates. Hold onto your root fingerprint, too; you'll need it to bootstrap your clients later.

Import the CA into the YubiKey

Now, let's import our PKI to the YubiKey.

```
$ ykman piv import-certificate 9a /mnt/ca/certs
/root_ca.crt
Successfully imported a new certificate.
$ ykman piv import-key 9a /mnt/ca/secrets
/root_ca_key
Enter PEM pass phrase: ...
Successfully imported a new private key.
$ ykman piv import-certificate 9c /mnt/ca/certs
/intermediate_ca.crt
Successfully imported a new certificate.
$ ykman piv import-key 9c /mnt/ca/secrets
/intermediate_ca_key
Enter PEM pass phrase: ...
```

```
Successfully imported a new private key.
$ ykman piv info
PIV version: 5.2.7
PIN tries remaining: 3
CHUID:
3019d4e739da739ced39ce739d836858210842108421c84210c3eb34104610300df33f7fd273e44f17361ce
CCC:    No data available.
Slot 9a:
        Algorithm:      ECCP256
        Subject DN:     CN=Tiny CA Root CA
        Issuer DN:      CN=Tiny CA Root CA
        Serial:
280998571002718115143415195266043025218
        Fingerprint:
d6b3b9ef79a42aeeabcd5580b2b516458ddb25d1af4ea7ff0845e624ec1bb609
        Not before:     2020-12-08 20:12:15
        Not after:      2030-12-08 20:12:15
Slot 9c:
        Algorithm:      ECCP256
        Subject DN:     CN=Tiny CA Intermediate CA
        Issuer DN:      CN=Tiny CA Root CA
        Serial:
383981404686758461431659830442976362 89
        Fingerprint:
fa21279c114ef44be899cb41e830b920faa6ce2c0ec5bc4f1c9310194e5837d2
        Not before:     2020-12-08 20:12:15
        Not after:      2030-12-08 20:12:15
```

OK! Now you'll copy out the CA certificate files, leave the private keys on the USB stick, and continue creating your CA.

```
$ sudo cp /mnt/ca/certs/intermediate_ca.crt
/mnt/ca/certs/root_ca.crt /root
$ cd
$ sudo umount /mnt
```

Finally, reconnect your CA to your local network to continue the setup.

Part 3: Configuring Your CA

You're going to re-run `step ca init` now, but *you're not going to use the certificates or keys that it generates*. You're just doing this to create the configuration files. The password you choose when prompted will be your *admin provisioner password*. Anyone who has it will be able to get any certificate from your CA, using the `step ca certificate` subcommand.

Don't use your root CA password for your provisioner, but pick something strong and store it somewhere safe.

```
$ sudo useradd step
$ sudo passwd -l step
$ sudo mkdir /etc/step-ca
$ export STEPPATH=/etc/step-ca
$ sudo --preserve-env step ca init --name="Tiny
CA" \
    --dns="tinyca.internal,10.20.30.42"
--address=":443" \
    --provisioner="you@example.com"
✔ What do you want your password to be? [leave
empty and we'll generate one]:

Generating root certificate...
all done!
```

```
Generating intermediate certificate...
all done!

✔ Root certificate: /etc/step-ca/certs
/root_ca.crt
✔ Root private key: /etc/step-ca/secrets
/root_ca_key
✔ Root fingerprint:
d52aa8dc57114fb28aafe0a4fa2795d3afeeb3a024bf6e6291077d99ff0cebc6
✔ Intermediate certificate: /etc/step-ca/certs
/intermediate_ca.crt
✔ Intermediate private key: /etc/step-ca/secrets
/intermediate_ca_key
✔ Database folder: /etc/step-ca/db
✔ Default configuration: /etc/step-ca/config
/defaults.json
✔ Certificate Authority configuration: /etc/step-
ca/config/ca.json

Your PKI is ready to go. To generate certificates
for individual services see 'step help ca'.

FEEDBACK 😍 🍻
      The step utility is not instrumented for
usage statistics. It does not
      phone home. But your feedback is extremely
valuable. Any information you
      can provide regarding how you're using
`step` helps. Please send us a
      sentence or two, good or bad:
feedback@smallstep.com or join
      https://github.com/smallstep/certificates
/discussions.
```

Now you'll add an ACME provisioner, which will turn your Tiny CA into a tiny Let's Encrypt.

```
$ sudo step ca provisioner add acme --type acme
--ca-config /etc/step-ca/config/ca.json
Success! Your `step-ca` config has been updated.
To pick up the new configuration SIGHUP (kill -1
<pid>) or restart the step-ca process.
```

Next, let's get your certificates in place.

```
$ sudo mv /root/root_ca.crt
/root/intermediate_ca.crt /etc/step-ca/certs
$ sudo rm -rf /etc/step-ca/secrets
```

Finally, you'll need to configure `step-ca` to use your YubiKey to sign certificates, using the intermediate key on the YubiKey. Notice that the default YubiKey PIN (`123456`) is shown here, too.

> You should change your YubiKey PIN, PUK, and management key if you haven't already! Learn how in this guide.

Now edit the file `/etc/step-ca/config/ca.json`. You'll want the top of the file to look like this:

```
{
        "root": "/etc/step-ca/certs/root_ca.crt",
        "federatedRoots": [],
        "crt": "/etc/step-ca/certs
/intermediate_ca.crt",
        "key": "yubikey:slot-id=9c",
        "kms": {
            "type": "yubikey",
```

```
            "pin": "123456"
        },
        "address": ":443",
...
```

Now you'll start up the CA and make sure it's running properly:

```
$ sudo chown -R step:step /etc/step-ca
$ sudo -u step step-ca /etc/step-ca/config/ca.json
2020/12/08 14:17:06 Serving HTTPS on :443 ...
```

In another window, you'll generate a test certificate for localhost.

This is where you'll need the CA fingerprint you created with your initial set of keys:

```
$ step ca bootstrap --ca-
url="https://tinyca.internal" --fingerprint
d6b3b9ef79a42aeeabcd5580b2b516458ddb25d1af4ea7ff0845e624ec1bb609
The root certificate has been saved in
/home/ubuntu/.step/certs/root_ca.crt.
Your configuration has been saved in /home/ubuntu
/.step/config/defaults.json.
$ step ca certificate "localhost" localhost.crt
localhost.key
✔ Provisioner: carl@smallstep.com (JWK) [kid:
Kn16nM7lnKBIsUSB-jd5wJDPgRsxzYsvilWTK4Rm2b0]
✔ Please enter the password to decrypt the
provisioner key:
✔ CA: https://tinyca.internal:443
✔ Certificate: localhost.crt
✔ Private Key: localhost.key
$ step certificate inspect localhost.crt --short
X.509v3 TLS Certificate (ECDSA P-256) [Serial:
2949...3005]
  Subject:     localhost
  Issuer:      Tiny Intermediate CA
  Provisioner: carl@smallstep.com [ID:
Kn16...m2b0]
  Valid from:  2020-12-08T22:18:34Z
          to:  2020-12-09T22:19:34Z
```

Great! You just signed your first X.509 TLS leaf certificate using the YubiKey and `step-ca`.

When you ask the CA to issue a leaf certificate for a TLS endpoint, you'll get a certificate file and an associated private key file. The certificate file will contain both the intermediate CA certificate and the leaf certificate you requested. This way, a device which trusts your root CA can verify the chain of trust from the root to the intermediate, and from the intermediate to the leaf.

Configure `systemd` to start the CA

In this section you'll set up a systemd service for `step-ca` so it starts when the system starts up.

You'll also configure systemd to stop the CA when the YubiKey is removed, and restart it when the YubiKey is reinserted.

First, you need to tell udev about your YubiKey by adding some udev rules, which will help make the YubiKey visible to systemd as a device.

```
$ sudo tee /etc/udev/rules.d/75-yubikey.rules >
/dev/null << EOF
ACTION=="add", SUBSYSTEM=="usb",
ENV{PRODUCT}=="1050/407/*", TAG+="systemd",
SYMLINK+="yubikey"
```

```
ACTION=="remove", SUBSYSTEM=="usb",
ENV{PRODUCT}=="1050/407/*", TAG+="systemd"
EOF
$ sudo udevadm control --reload-rules
```

Here, the format of the `ENV{PRODUCT}` value is `{vendorId}/{productId}/*`. Yubico's vendor ID is `1050`, and `407` is the product ID for the YubiKey 5 NFC. If you're using a different YubiKey, you can find your model number here.

Now you'll set up the CA as a systemd service that will:

- run on system startup, when the YubiKey is inserted
- stop when the YubiKey is removed
- start again when the YubiKey is reinserted

```
$ sudo tee /etc/systemd/system/step-ca.service >
/dev/null << EOF
[Unit]
Description=step-ca
BindsTo=dev-yubikey.device
After=dev-yubikey.device

[Service]
User=step
Group=step
ExecStart=/bin/sh -c '/usr/local/bin/step-ca
/etc/step-ca/config/ca.json'
Type=simple
Restart=on-failure
RestartSec=10

[Install]
WantedBy=multi-user.target
EOF
$ sudo mkdir /etc/systemd/system/dev-
yubikey.device.wants
$ sudo ln -s /etc/systemd/system/step-ca.service
/etc/systemd/system/dev-yubikey.device.wants/
$ sudo systemctl daemon-reload
$ sudo systemctl enable step-ca
```

Now insert the YubiKey and the service should start:

```
$ sudo systemctl status step-ca
● step-ca.service - step-ca
     Loaded: loaded (/etc/systemd/system/step-
ca.service; enabled; vendor preset: enabled)
     Active: active (running) since Tue 2020-12-08
14:27:02 PST; 3s ago
   Main PID: 3269 (sh)
      Tasks: 9 (limit: 2099)
     CGroup: /system.slice/step-ca.service
             ├─3269 /bin/sh -c /usr/local
/bin/step-ca /etc/step-ca/config/ca.json
             └─3270 /usr/local/bin/step-ca
/etc/step-ca/config/ca.json

Dec 08 14:27:02 tinyca systemd[1]: Started step-
ca.
Dec 08 14:27:02 tinyca sh[3270]: 2020/12/08
14:27:02 Serving HTTPS on :443 ...
```

Now restart your system and ensure that the CA starts up automatically.

Test out removing the YubiKey, and you should see that the CA

stops.

Reinsert it, and the CA should start up again.

Finally, turn on the firewall and disable SSH access

Your tiny CA will be most secure without any SSH access at all. The only open port will be 443, for the CA. For maintenance, you'll need to plug in a keyboard and a display.

```
$ sudo tee /etc/ufw/applications.d/step-ca-server
> /dev/null << EOF
[step-ca]
title=Smallstep CA
description=step-ca is an online X.509 and SSH
Certificate Authority
ports=443/tcp
EOF
$ sudo ufw allow step-ca
$ sudo ufw enable
Command may disrupt existing ssh connections.
Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
```

Using Your CA

You did it! Your CA is up and running.

Bootstrapping a new device into your PKI

When you run `step ca bootstrap` (as above) on a new device, the root certificate `root_ca.crt` is downloaded from the CA. If you run `step ca bootstrap --install --ca-url=https://your.ca --fingerprint=your-ca-fingerprint`, it will install the root certificate into your device's trust store.

You can also use the `step` command for easy installation of your root CA certificate (`step certificate install`), for ACME enrollment (`step ca certificate example.com example.crt example.key --provisioner acme`) and for renewal of any certificate that hasn't yet expired (`step ca renew example.crt example.key`).

For mobile devices, you can usually install a certificate by sending it to yourself via Bluetooth or AirDrop, or as an email attachment. Make sure the certificate isn't just installed, but actually trusted by the device. This usually involves a couple of confirmation steps on the device.

Use ACME!

With the ACME provisioner, you can use software like Certbot or LEGO CLI to easily get and renew certificates for any endpoint. Our tutorials on running a private ACME server and configuring popular ACME clients to use a private ACME server will show you how to get ACME certificates from your CA using the most common ACME clients and ACME-supporting services.

Automating certificate renewal

Because certificates from your CA have a 24-hour lifetime, you'll want to renew them every 16ish hours. Our renewal documentation has a few options for setting up renewal on your clients.

Further Reading

Now that you have an internal CA, here's a few useful resources:

- To get more familiar with the `step` command and how it interfaces with your CA, try out some of the examples in Basic Certificate Authority Operations.
- Hello mTLS shows you how to get mutual TLS authentication configured for several common services and programming languages, using the `step` command.
- There's also a lot to learn about the different provisioners you can add to your CA to suit your workflows. See Configuring `step-ca`.
- Bonus: Want to use SSH certificates? You can turn your tiny CA into an SSH CA, and use certificates and single sign-on for your SSH hosts. We have a blog post and video walk-through that describes how to set it up.